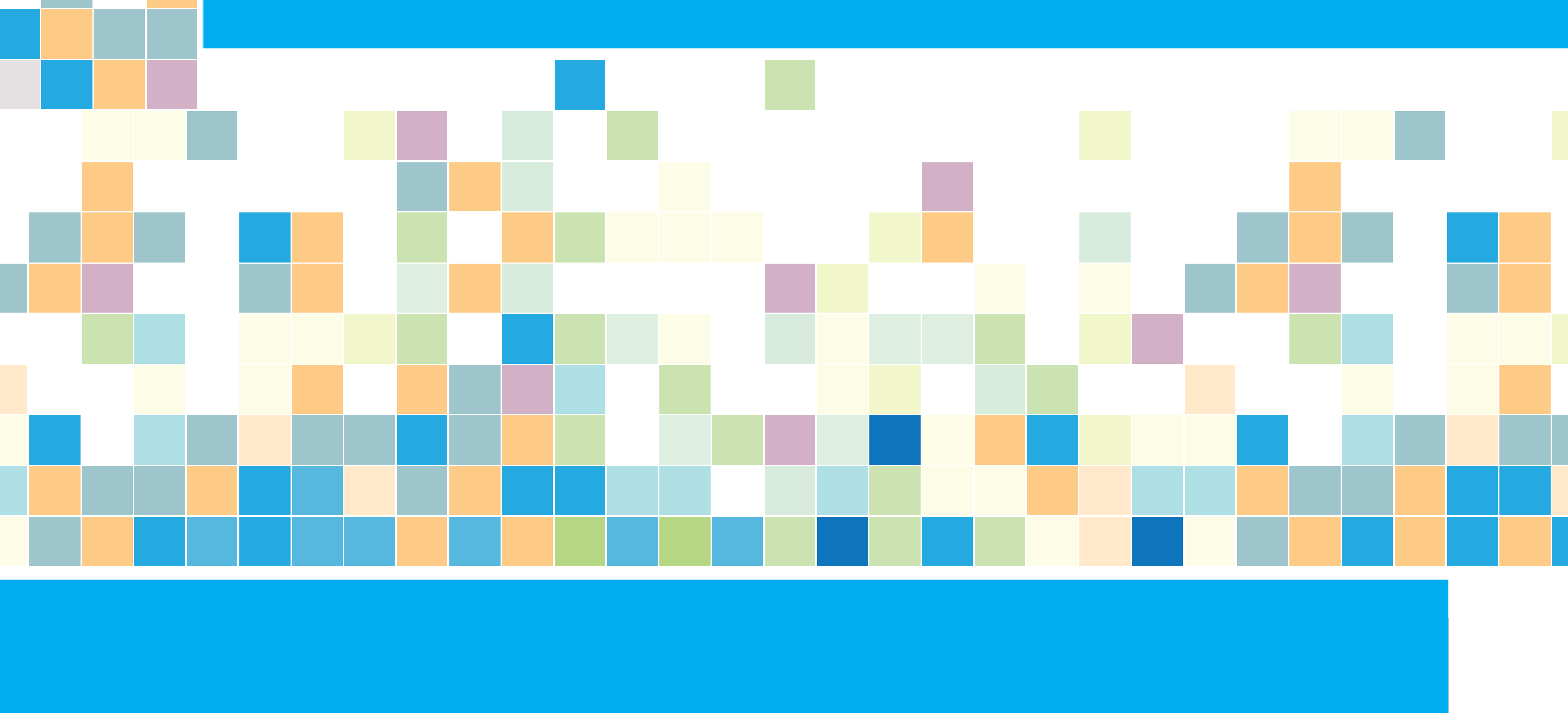
21世纪高等学校计算机教育实用规划教材

PHP Zend Framework 项目开发基础案例教程

马石安 魏文平 编著



清华大学出版社

21世纪高等学校计算机教育实用规划教材

PHP Zend Framework 项目开发基础案例教程

马石安　魏文平　编著

清华大学出版社
北　京

内 容 简 介

本书以实际项目——XX 办公自动化管理系统的开发——为案例，详细介绍使用 PHP 的企业级框架 Zend Framework 进行 Web 应用开发的步骤及关键技术。全书共 12 章，系统地介绍 Zend Framework 基本结构、运行原理、开发环境配置，以及 Zend_Navigation 导航、Zend_Form 表单、Zend_Db 数据库、Zend_Acl 访问控制、Zend_Cache 缓存等组件技术。

本书是一本 Zend Framework 的入门级实例教程，适合具备 PHP Web 应用开发基础、希望学习框架技术来提升开发能力的读者，以及掌握了基本的 Web 编程语言和面向对象技术，但还没有太多项目开发经验的高校在校学生。本书可作为高等院校计算机科学专业、各类网络技术培训的教材使用，也可供软件开发人员进行项目开发、在校学生进行课程设计与毕业设计时参考。

图书在版编目(CIP)数据

PHP Zend Framework 项目开发基础案例教程/马石安，魏文平编著. --北京：清华大学出版社，2015
21 世纪高等学校计算机教育实用规划教材
ISBN 978-7-302-40561-0

Ⅰ. ①P…　Ⅱ. ①马…　②魏…　Ⅲ. ①PHP 语言－程序设计　Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字(2015)第 145847 号

责任编辑： 魏江江　王冰飞
封面设计：
责任校对： 焦丽丽
责任印制：

出版发行： 清华大学出版社
网　　址： http://www.tup.com.cn，http://www.wqbook.com
地　　址： 北京清华大学学研大厦 A 座　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载： http://www.tup.com.cn，010-62795954
印 刷 者：
装 订 者：
经　　销： 全国新华书店
开　　本： 185mm×260mm　　**印　　张：** 19　　**字　　数：** 463 千字
版　　次： 2015 年 11 月第 1 版　　**印　　次：** 2015 年 11 月第1 次印刷
印　　数： 1～　000
定　　价： .00 元

产品编号：062534-01

出版说明

随着我国高等教育规模的扩大以及产业结构调整的进一步完善，社会对高层次应用型人才的需求将更加迫切。各地高校紧密结合地方经济建设发展需要，科学运用市场调节机制，合理调整和配置教育资源，在改革和改造传统学科专业的基础上，加强工程型和应用型学科专业建设，积极设置主要面向地方支柱产业、高新技术产业、服务业的工程型和应用型学科专业，积极为地方经济建设输送各类应用型人才。各高校加大了使用信息科学等现代科学技术提升、改造传统学科专业的力度，从而实现传统学科专业向工程型和应用型学科专业的发展与转变。在发挥传统学科专业师资力量强、办学经验丰富、教学资源充裕等优势的同时，不断更新教学内容、改革课程体系，使工程型和应用型学科专业教育与经济建设相适应。计算机课程教学在从传统学科向工程型和应用型学科转变中起着至关重要的作用，工程型和应用型学科专业中的计算机课程设置、内容体系和教学手段及方法等也具有不同于传统学科的鲜明特点。

为了配合高校工程型和应用型学科专业的建设和发展，急需出版一批内容新、体系新、方法新、手段新的高水平计算机课程教材。目前，工程型和应用型学科专业计算机课程教材的建设工作仍滞后于教学改革的实践，如现有的计算机教材中有不少内容陈旧（依然用传统专业计算机教材代替工程型和应用型学科专业教材），重理论、轻实践，不能满足新的教学计划、课程设置的需要；一些课程的教材可供选择的品种太少；一些基础课的教材虽然品种较多，但低水平重复严重；有些教材内容庞杂，书越编越厚；专业课教材、教学辅助教材及教学参考书短缺，等等，都不利于学生能力的提高和素质的培养。为此，在教育部相关教学指导委员会专家的指导和建议下，清华大学出版社组织出版本系列教材，以满足工程型和应用型学科专业计算机课程教学的需要。本系列教材在规划过程中体现了如下一些基本原则和特点。

（1）面向工程型与应用型学科专业，强调计算机在各专业中的应用。教材内容坚持基本理论适度，反映基本理论和原理的综合应用，强调实践和应用环节。

（2）反映教学需要，促进教学发展。教材规划以新的工程型和应用型专业目录为依据。教材要适应多样化的教学需要，正确把握教学内容和课程体系的改革方向，在选择教材内容和编写体系时注意体现素质教育、创新能力与实践能力的培养，为学生知识、能力、素质协调发展创造条件。

（3）实施精品战略，突出重点，保证质量。规划教材建设仍然把重点放在公共基础课和专业基础课的教材建设上；特别注意选择并安排一部分原来基础比较好的优秀教材或讲义修订再版，逐步形成精品教材；提倡并鼓励编写体现工程型和应用型专业教学内容和课程体系改革成果的教材。

(4) 主张一纲多本,合理配套。基础课和专业基础课教材要配套,同一门课程可以有多本具有不同内容特点的教材。处理好教材统一性与多样化,基本教材与辅助教材,教学参考书,文字教材与软件教材的关系,实现教材系列资源配套。

(5) 依靠专家,择优选用。在制订教材规划时要依靠各课程专家在调查研究本课程教材建设现状的基础上提出规划选题。在落实主编人选时,要引入竞争机制,通过申报、评审确定主编。书稿完成后要认真实行审稿程序,确保出书质量。

繁荣教材出版事业,提高教材质量的关键是教师。建立一支高水平的以老带新的教材编写队伍才能保证教材的编写质量和建设力度,希望有志于教材建设的教师能够加入到我们的编写队伍中来。

21 世纪高等学校计算机教育实用规划教材编委会
联系人:魏江江 weijj@tup.tsinghua.edu.cn



前　言

PHP语言是目前国内外最普及、使用最广泛的互联网开发语言之一，它不仅具有功能丰富、表达能力强、使用方便灵活、执行效率高、可移植性好等优点，而且具有开放的源代码、多数据库支持、面向对象支持、容易学习、完全免费等特点，越来越受欢迎，正在逐渐成为Web应用开发的主流语言。Zend Framework作为PHP的官方框架，充分发挥了PHP语言的特点，所采用的面向对象技术、前端控制技术以及MVC程序设计模式更是极大地提高了Web项目的开发效率，并且使应用程序的扩展与维护变得异常简单与容易，是大、中型Web应用的理想开发框架。

本书的写作背景基于以下两个方面：一是专业教育的需要；二是实际项目的驱使。目前进行的高等教育课程改革更加关注学生综合应用能力的培养以及课程设置与市场需要的匹配。因此，在高校计算机科学技术专业教育中新增了大量的实践与选修课程，用于培养学生综合应用基础理论知识的能力，我们在教学过程中深深感到了这方面教材的匮乏。另外，作者在教学之余还主持并参与了多项软件产品的研发工作。本书案例项目就是来自于实际项目——XX办公自动化管理系统，是其简化版本。该项目针对某大学的一所独立学院，使用了目前Web应用开发中应用最广的PHP Zend Framework框架等网络编程的主流技术，把该项目作为一个教学案例，无论是从技术性、综合性，还是从规范性、完整性方面都是非常合适的。目前，PHP虽然正在逐渐成为Web应用开发的主流语言，但高校的计算机专业教学体系中却很少设置相关的课程，本书的出版会弥补这方面的不足。

本书案例项目实现了办公自动化管理系统的常用功能，包括用户信息管理、访问控制管理、公文信息管理、事务信息管理、新闻资讯管理、留言信息管理以及公共服务管理中的用户网络空间、日程安排、工作日志、数据备份、系统日志等。本书以这些系统功能的实现过程为依托，由浅入深、循序渐进地介绍了Zend Framework框架的基本结构、运行原理、开发环境配置以及Zend_Navigation导航、Zend_Form表单、Zend_Db数据库、Zend_Acl访问控制、Zend_Cache缓存等组件技术。全书共分12章，内容如下：

第1章主要介绍采用PHP的Zend Framework框架进行项目开发前的一些准备工作，包括对Zend Framework框架的简单认识、开发环境的搭建与配置、常用开发工具Zend Studio和Notepad++的使用等。

第2章详细介绍Zend Framework框架的基本结构及运行原理，并对框架的几个重要文件进行简单的分析。本章内容是项目开发的理论基础，其中的MVC机制更是后续学习

的主线。

第 3 章是系统概述及总体设计。本章从软件工程的角度简要地介绍了 Web 应用开发的一般步骤,包括系统分析、系统设计、数据库设计以及公共文件设计等内容。

第 4 章介绍系统的页面设计方法及 Zend Framework 的布局模式,主要内容包括系统初始设置、CSS 样式概述、典型页面设计以及 Zend_Layout 布局模板。

第 5 章介绍 Zend Framework 的导航及数据库操作,通过系统新闻资讯管理模块的实现详细讲解 Zend Framework 框架的 Zend_Navigation 及 Zend_Db 组件技术。

第 6 章通过用户登录与注册功能的实现,介绍 Zend Framework 表单的创建、设置及处理技术。Zend_Form 表单与传统的 HTML 表单相比,具有无与伦比的技术优势,是实现人机交互、保证信息安全、减少代码冗余的最佳技术方案。

第 7 章主要介绍 Zend Framework 框架的模块技术以及数据库的增、删、改、查等详细操作方法。本章既是对第 5 章内容的进一步阐述与扩充,也是对前面其他章节知识的一个实际应用。

第 8 章介绍公文信息管理功能的实现,包括公文的添加、分类检索、状态跟踪等。该功能是办公自动化管理系统的核心功能,涉及的内容繁杂,业务逻辑也相对复杂。

第 9 章介绍留言信息管理功能的实现,内部留言系统是应用系统内部用户之间短暂交流与沟通的平台,它综合了常用即时通信工具及邮件系统的特点,是办公自动化管理系统中不可或缺的功能模块。

第 10 章介绍事务信息管理功能的实现,包括数据库的设计、事务的添加、事务的分类显示、事务提醒、批示回复以及事务处理状态跟踪等。本章的事务添加采用了 Zend Framework 的视图助手,它是一种 Web 表单创建的新的解决方案。

第 11 章介绍办公自动化管理系统中常用办公功能的实现,包括用户网络空间、日程安排的创建与管理。用户网络空间类似于一个简单的资源管理器,能够进行文件夹的创建与删除、文件的复制与粘贴等基本功能。

第 12 章实现办公自动化管理系统的访问控制、缓存等 Web 应用的安全、性能优化技术方案,并对系统进行了简要的完善,包括图形验证码、系统日志、数据备份等。

本书的主要特色如下:

(1) 技术先进,使用广泛。

本书介绍的 Zend Framework 框架技术在目前网络应用开发中被广泛使用。它作为 PHP 网络应用开发的官方框架,拥有非常庞大的用户基础和国际顶级的开发人员支持,发展稳定、前景良好。

(2) 案例完整、实用性强。

本书案例是一个实际项目的简化版本,稍做改进与完善即可应用到实际项目的开发中。

(3) 内容翔实,循序渐进。

本书紧紧围绕应用实例,从软件工程的角度出发,按照项目开发的顺序全面地介绍程序开发规范及流程,使读者在很短的时间内即可掌握 Web 应用开发的步骤与常用技术。

（4）重点突出，难点分散。

本书以介绍 Zend Framework 框架技术为重点，主要介绍应用的业务处理逻辑的实现，对页面表现技术，例如 CSS 样式、JavaScript、JQuery 等技术进行了略化处理。每章突出一个技术难点，每种技术的介绍均以从应用到原理的顺序展开，让读者先看到效果，然后激发其探究“为什么”的兴趣。

（5）由浅入深，前后呼应。

Web 应用的开发是一个基础理论知识的综合应用过程，会涉及很多方面。本书实例功能的实现采用了由浅入深、逐步完善的方式，将技术难点分散于各个章节中，做到了叙述上的前后呼应、技术上的逐步加深。

（6）资源丰富，探讨方便。

为帮助读者学习，本书除提供源码及相关工具的下载（清华大学出版社网站 http://www.tup.tsinghua.edu.cn）外，还创建了技术支持网站 http://www.wmstudio.net.cn，在这里读者不仅可以看到案例的运行效果，还可以与作者进行交流，并对相关问题进行探讨。

本书是一本 Zend Framework 框架的入门级实例教程，适合具备 PHP Web 应用开发基础、希望学习框架技术来提升开发能力的读者，尤其适合掌握了基本的 Web 编程语言和面向对象技术，但还没有太多项目开发经验的高校在校学生。本书可作为高等院校计算机专业实践及选修课程、课程设计、毕业设计、计算机专业培训、计算机程序设计竞赛的教材及参考书籍，同时也可供软件开发人员进行项目开发时参考。

本书第 1～7 章由马石安编写，第 8～12 章由魏文平编写，所有图片的配置、代码的测试由魏文平完成。全书由马石安统一修改、整理和定稿。

在编写本书的过程中，参考和引用了大量的书籍、文献以及网络中的技术资料，在此向这些文献的作者表示衷心感谢。另外，江汉大学、清华大学出版社的领导及各位同仁对本书的编著、出版给予了大力支持与帮助，在此一并表示感谢。由于作者水平有限、加之时间仓促，书中难免存在缺点与疏漏之处，敬请广大师生、读者批评指正。作者的联系方式为 wenpingwei@163.com。

编　者

2015 年 8 月

目　　录

第 1 章　Zend Framework 开发环境

Zend Framework 是由 Zend Technologies 公司支持开发的、完全基于 PHP 5 的开源 PHP 项目开发框架，可用于 Web 应用与服务的开发。作为 PHP 语言的标准应用程序框架，Zend Framework 采用 MVC 的架构模式，将应用中的业务逻辑与视图进行分离，方便了程序的开发、扩展与维护。

本章介绍 Zend Framework 项目开发前的一些准备工作，包括开发环境的配置、常用开发工具的使用以及各种必备的技术资料。

1.1　Zend Framework 概述

应用程序框架顾名思义就是应用程序的基本结构，也指应用程序运行的基本架构类库。它是一种相对固定的设计模式，也是一种可重用的、半完成的应用程序。使用框架进行项目开发，可以让开发者把注意力更多地集中到应用的整体结构和业务逻辑上，从而提高项目开发的效率与质量。

Zend Framework 是用 PHP 5.3 及更高版本开发 Web 应用和服务的开源框架，作为一种官方的、热门的 PHP 框架技术，它在目前的 PHP 项目开发领域受到了广泛的关注与青睐。Zend Framework 框架采用纯面向对象技术编码实现，以组件的形式进行类的组合。其组件结构也与其他的框架不同，采用了松耦合形式，每个组件几乎不依靠其他组件。这样，组件既可以联合使用也可以独立使用，充分发挥了面向对象技术的代码重用功能，减少了程序的冗余，使应用程序更加强壮与稳健。

1.1.1　Zend Framework 的特点

1. 开发公司

Zend Technologies 公司是一家互联网基础架构软件公司，成立于 1999 年 11 月，由 Zeev Suraski 和 Andi Gutmans 两位开源 PHP 的缔造者共同创建。该公司总部设在美国加利福尼亚州的库比蒂诺，技术中心设在以色列特拉维夫的拉马特甘，并在法国、意大利、德国等多个国家设有办事机构。

Zend Technologies 公司的投资商是包括 Azure Capital Partners、Index Ventures 和 Platinum Venture Capita 在内的非常有实力的投资公司。更为重要的是，在 IT 行业领先的 SAP 和 Intel 也是 Zend 公司的投资商。目前，Zend 公司已经得到了来自 IBM、Oracle、Sun、Microsoft 公司的大力支持，并与 IBM、Oracle 达成了战略合作伙伴关系，共同推动 PHP 技术的发展。

Zend Technologies 公司是 PHP 运行引擎 Zend Engine 的开发商，它的两个奠基人 Andi Gutmans 和 Zeev Suraski 与其他以色列程序员一起发展了由 Rasmus Lerdorf 开创的 PHP 语言。由于具有国际性的技术权威，Zend 公司和他的创建者在 PHP 应用开发以及开源团体中持续处于领导的核心地位，对于 PHP 技术的迅猛发展起到了强有力的推动作用，并且为企业和个人持续提供有关 PHP 应用的、国际领先的、专业的技术解决方案。

2. Zend Framework 的优点

Zend Framework 作为 PHP 的官方框架，不仅拥有庞大的用户基础，而且有对 PHP 内部构造非常熟悉的、世界顶级的开发者的支持，因此与其他 PHP 框架相比，它具有更高的信赖度、更稳定的发展前景。另外，从 Zend 公司对 Zend Framework 的投入来看，中断对 Zend Framework 的支持是不太可能的，将来一定会继续对 Zend Framework 版本进行升级与补丁修正，并提供良好的技术支持。从技术的角度来说，Zend Framework 还具有如下特点：

1）基于 PHP 建立

Zend Framework 是采用 PHP 编写的一套集成开发框架，可以说是从 PHP 中来，到 PHP 中去，因而与运行环境具有极好的融合性，在所有 PHP 框架里，它的功能强大，而且稳健。

2）面向对象

PHP 支持面向对象的开发形式，特别是 PHP 5 对类和对象提供了更为有效的支持。Zend Framework 是纯正的 OOP 框架，代码严谨、规范，对 PHP 的侵入性要比其他框架低，适合多人联合开发。

3）使用 MVC 模式

MVC（模型—视图—控制器）开发模式是一种新兴的 Web 应用程序开发方式，使用此种方式可以实现 Web 页面表现层与逻辑层的分离。这样，应用程序结构层次清晰，维护、扩展方便。

4）使用松耦合设计

Zend Framework 采用松耦合（Use-at-will）设计，每个组件几乎不依赖其他组件，这样可以让开发者独立使用组件，扩展了组件的使用范围。

5）多数据库支持

PDO（PHP Data Objects）扩展为 PHP 访问数据库定义了一个轻量级的、一致性的接口，它提供了一个数据访问抽象层，这样无论使用什么数据库都可以通过一致的函数执行查询和获取数据。由于使用了 PDO 扩展，Zend Framework 支持多种数据库，例如 MySQL、Oracle、IBM DB2、Microsoft SQL Server、PostgreSQL、SQLite 和 Informix Dynamic Server 等。

6）多邮件系统支持

Zend Framework 支持多种邮件收发系统，例如 MBox、Maildir、POP3 和 IMAP4 等。

7）灵活的缓存机制

Zend Framework 具有非常灵活的缓存机制，支持多种缓存方式，可以将缓存写入内存、文件或数据库等。

8）具有开放源代码贡献者

Zend Framework 秉承了开放源代码的特点，而且有 Zend 公司作为领导者的大量 PHP

程序员参与了此项工程的开发，这样，Zend Framework 的功能会越来越强大，性能也会越来越稳定。

9）由贡献者负责其代码的知识产权

随着信息技术的发展，知识产权越来越受到人们的重视。Zend Framework 作为由贡献者参与的开源程序，参与项目开发的所有贡献者保证他们所使用的代码的自主知识产权。

10）宽松的认证方式

Zend Framework 采用 New BSD License 的认证方式。New BSD License 认证是只要使用者在代码中注明著作权就可以自由发布的一种认证方法，同时也是所有认证方式中限制最少的一种。Zend Framework 框架的源码都是独立编程的，不存在 Zend 公司以外的著作权拥有者，因为应用 Zend Framework 构筑的系统在发布时没有任何限制、商用的情况下也不必担心侵权的问题。

3. Zend Framework 的缺点

Zend Framework 框架虽然拥有巨大的技术优势，但这并不表示它是 PHP 应用开发的唯一选择。任何事物都不是完美的，Zend Framework 也有很明显的缺陷。

1）Zend Framework 源文件多、体积大

由于 Zend Framework 功能强大，是一种重量级的企业框架，所以它的源文件非常多，下载压缩包的体积也比较大。这与某些轻量级的框架相比，Zend Framework 显得有些臃肿。

其实在下载的压缩包里有许多文件并不属于框架本身，例如示例文件、帮助文档以及功能扩展文件等。另外，Zend Framework 提供精简版的下载，也可以在开发中根据系统需求删除某些不用的组件。

2）Zend Framework 执行速度慢

和其他大型 Web 框架类似，Zend Framework 有一个非常庞大的前端控制器（Front Controller）。但由于 PHP 运行时环境的特殊性，即每次请求都是独立的上下文，这个前端控制器不得不在每次请求时都被重新初始化一次。这样的运行方式给性能带来了非常大的开销，这也被认为是 Zend Framework 的性能瓶颈所在。另外，与前端控制器运行方式类似，在 Zend Framework 的数据库操作中，通过 Zend_Db 获取数据库中表结构信息的操作，也是每次请求都要重复进行的动作。

事实上，Zend_Db 可以通过 Memcached、Apc 等类似的外部缓存器来缓存表结构，但前端控制器设计的复杂确实不是缓存可以解决的。这并不说明 Zend Framework 设计有问题，而是说明并不是所有的项目、应用都适合使用 Zend Framework 框架，要靠项目开发者针对自身情况权衡。

Zend Framework 的执行速度在新的 Zend Framework 2 中得到了很大的改善，相信随着信息技术的发展、服务器及网络传输性能的不断提高，这些问题都将会迎刃而解。

3）学习资料少，入门比较难

Zend Framework 采用纯正的 OOP 技术，要求学习者具有扎实的基础理论功底，这使得不少学习者望而生畏。另外，Zend Framework 的学习文档大而全，且不怎么友好，显得太理论化，难以理解，与实际应用之间的差距较大。尤其让学习者烦恼的是，关于 Zend Framework 的中文教程奇缺，官方给出的中文手册仅仅是一个简单的英汉对照手册，不符合中国人的学习习惯。这些都使学习者学习 Zend Framework 变得更加困难。

基于此,我们将一个实际的 Zend Framework 项目简单化、步骤化后写成了本书,希望给大家的学习提供一些帮助。

1.1.2 Zend Framework 的常用组件

作为一整套 PHP 开发框架的解决方案,Zend Framework 分为若干个组成部分。每个部分都称为一个组件,每个组件实现特定的功能。按照功能不同,可以把 Zend Framework 划分为以下 5 个组成部分。

1. MVC 组件

MVC 组件用于实现 MVC 开发模式的各个部分,包括控制器、视图等,其中的 Zend_Controller 是 Zend Framework 架构的控制核心。

2. 功能组件

功能组件作为 Zend Framework 的功能核心,不仅为 Zend Framework 框架服务,还适用于其他各类应用程序,可以为应用程序提供功能各异的支持,包括文件与类的动态加载、配置数据的使用、过滤与校验、数据缓存、访问控制、邮件操作、系统日志等。

3. 数据操作组件

数据操作组件用于数据操作,其中包括数据库操作、文本搜索以及 PDF 操作等。文本搜索引擎可以对文件系统进行操作,并完成相应的功能,例如新建文件夹、上传文件等。

4. 服务类组件

服务类组件用于实现各类服务,例如 RSS、XML-RPC、REST 等。

5. 国际化组件

国际化组件用于实现应用程序的国际化功能,也就是要让同样的应用程序能够在不同的地方使用。这里需要解决的问题主要有语言、日期与时间、货币、计量单位以及数据格式等。

需要强调的是,Zend Framework 一方面是以 MVC 组件中的 Zend_Controller 前端控制器为中心的软件架构产品,另一方面也是数据库连接、认证、过滤器等各种功能组件的类库。这些功能组件彼此独立,都能以单独的类库形式被使用。例如,在不需要框架的项目中,如果只需要使用数据库操作功能,只需要将 Zend Framework 的 Zend_Db 组件类库导入到项目中就可以了,这也体现了 Zend Framework 的松耦合性设计。

Zend Framework 常用组件及其功能如表 1.1 所示。

表 1.1 Zend Framework 常用组件及功能

组　　件	功　　能
Zend_Acl	实现访问控制列表功能
Zend_Auth	提供认证功能
Zend_Cache	实现数据缓存功能
Zend_Config	配置文件管理,可以从 INI 或者 XML 配置文件中读取数据
Zend_Controller	MVC 中的控制器部分,为应用程序提供全面的控制,它将请求转化为特定的行为并确保其执行
Zend_Currency	操作货币/数值的格式
Zend_Date	实现不同地域的日期时间功能

续表

组　　件	功　　能
Zend_Db	基于 PHP 数据对象(PDO),并提供一种通用的方式来访问数据库
Zend_Debug	提供调试功能
Zend_Exception	实现异常处理功能
Zend_Feed	提供连接 RSS/Atom Feed 的手段
Zend_Filter	提供对输入字符串进行过滤的方法
Zend_Form	制作复杂的 Form 表单
Zend_Http	控制经由 HTTP 协议的通信
Zend_Json	实现将 PHP 对象转换成 JavaScript 对象符号,或者进行反方向的转换
Zend_Loader	动态加载文件或类
Zend_Locale	其他国际化类的基类,为其他国际化类提供支持
Zend_Log	提供通用日志功能
Zend_Mail	提供邮件收发功能
Zend_Measure	实现不同地域计量单位的转换
Zend_Memory	实现在限制内存环境下的操作
Zend_Mime	用于为 Zend_Mail 组件解码 MIME 消息
Zend_Pdf	创建新的 PDF 文档,以及加载和编辑现有文档
Zend_Rest	实现 REST 服务
Zend_Registry	值与对象(全局变量)的存储容器
Zend_Search_Lucene	用来构建基于文本的全文搜索引擎
Zend_Servic	使用网络上知名服务提供商的 API
Zend_Session	管理、操作 Session 数据
Zend_Translate	翻译组件,可以通过适配器实现不同语言的互译
Zend_Uri	用于对 URI 进行操作
Zend_Validate	实现校验器功能,判断某个字符串是否符合某个标准
Zend_View	处理 MVC 模式的“视图”部分
Zend_XmlRpc	提供连接 XML-RPC 式服务的手段

1.2　搭建开发环境

进行 Zend Framework 项目开发,首先需要搭建相应的开发环境。目前,PHP 应用的开发环境普遍采用 XAMP 的平台形式。所谓 XAMP,其实是几种软件的组合,其中的 X 是指操作系统,例如 L(Linux)、W(Windows)、M(Mac OS)等,而 A、M 和 P 分别代表 Apache 网络服务器、MySQL 数据库管理系统和 PHP(或者 Peri)应用服务器。PHP、Apache 及 MySQL 是经典的 Web 应用开发平台。

本书为 Zend Framework 项目开发基础入门教程,考虑到大多数学习者使用的是 Windows 操作系统,所以采用 WAMP 的开发环境配置方案。

1.2.1　集成软件包的安装与配置

在 Windows 操作系统中分别安装并配置 Apache、MySQL 及 PHP 会非常复杂,这里使用集成软件包自动完成 WAMP 开发环境的搭建。

目前广泛使用的集成软件包有两个：一个是 XAMPP，另一个是 AppServ。下面对这两个集成软件包的安装与配置进行简单介绍。

1. XAMPP 软件包

XAMPP 是一个功能强大的 PHP 开发集成软件包，是一款完全免费的 Apache 发行版本，包括 MySQL、PHP 和 Perl。该软件包原来的名字叫 LAMPP，为了避免误解，最新的几个版本改名为 XAMPP。目前，XAMPP 提供 4 种不同的版本，可以分别在 Windows、Linux、Mac OS、Solaris 4 种操作系统下安装使用，支持英文、简体中文、繁体中文、韩文、俄文、日文等多种语言。其官方网址为 http://www.apachefriends.org，提供的下载格式有可执行文件和压缩包两种。

本书所用的 Windows 操作系统为 Windows 7 旗舰版，采用的 XAMPP 软件包是目前最新的 1.8.3 版本，下载格式为可执行文件形式。当使用的版本不同时，安装方法、文件路径等可能会有不同，请大家注意。

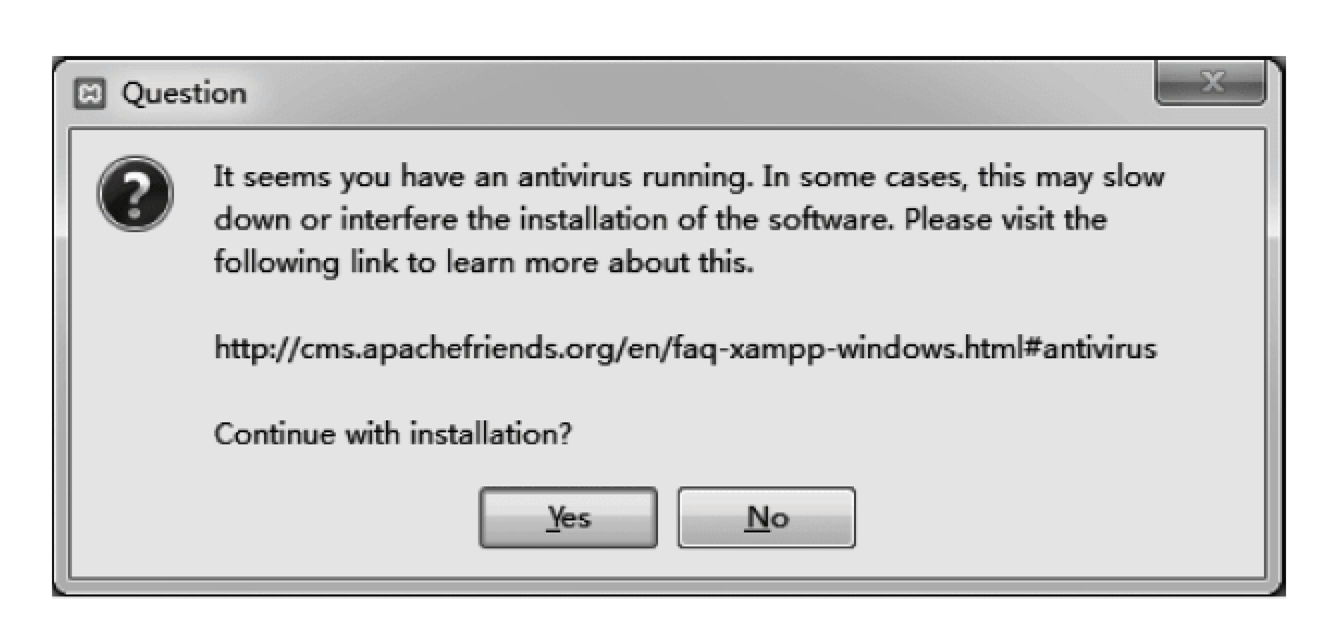
XAMPP 集成软件包的下载与安装步骤如下：

（1）在计算机上创建 XAMPP 资源文件夹，例如 E:\XAMPP。

（2）打开浏览器，在地址栏中输入 http://www.apachefriends.org，单击页面中的 XAMPP for Windows 链接，下载 XAMPP 的 Windows 版本，这里为 xampp-win32-1.8.3-4-VC11-installer.exe，将其保存在 E:\XAMPP 文件夹中。

（3）双击下载的 XAMPP for Windows 安装文件，启动安装程序。

如果是 Windows 7 操作系统，会连续弹出如图 1.1 所示的两个提示框，直接单击 Yes 和 OK 按钮进入下一步。



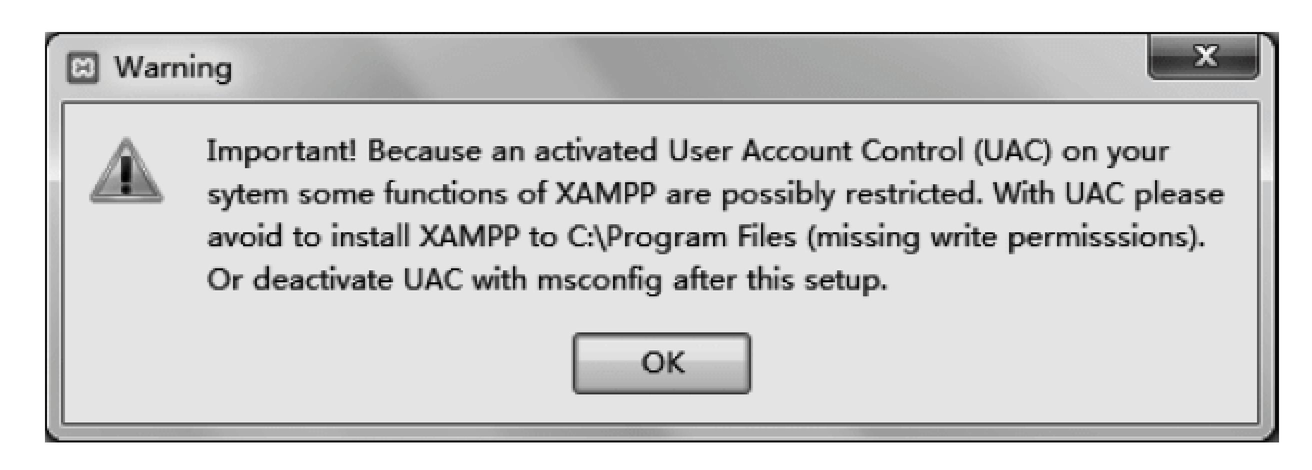


图 1.1　安装提示对话框

（4）在接下来的对话框中连续单击 Next 按钮，直到弹出如图 1.2 所示的正式安装界面。

（5）单击图 1.2 所示对话框中的 Next 按钮，打开如图 1.3 所示的界面。

这一步选择需要安装的组件，请务必择选 Apache、MySQL 两个组件，用于安装 Apache Web 服务器以及 MySQL 数据库服务器，建议选择 phpMyAdmin 数据库管理工具，以方便

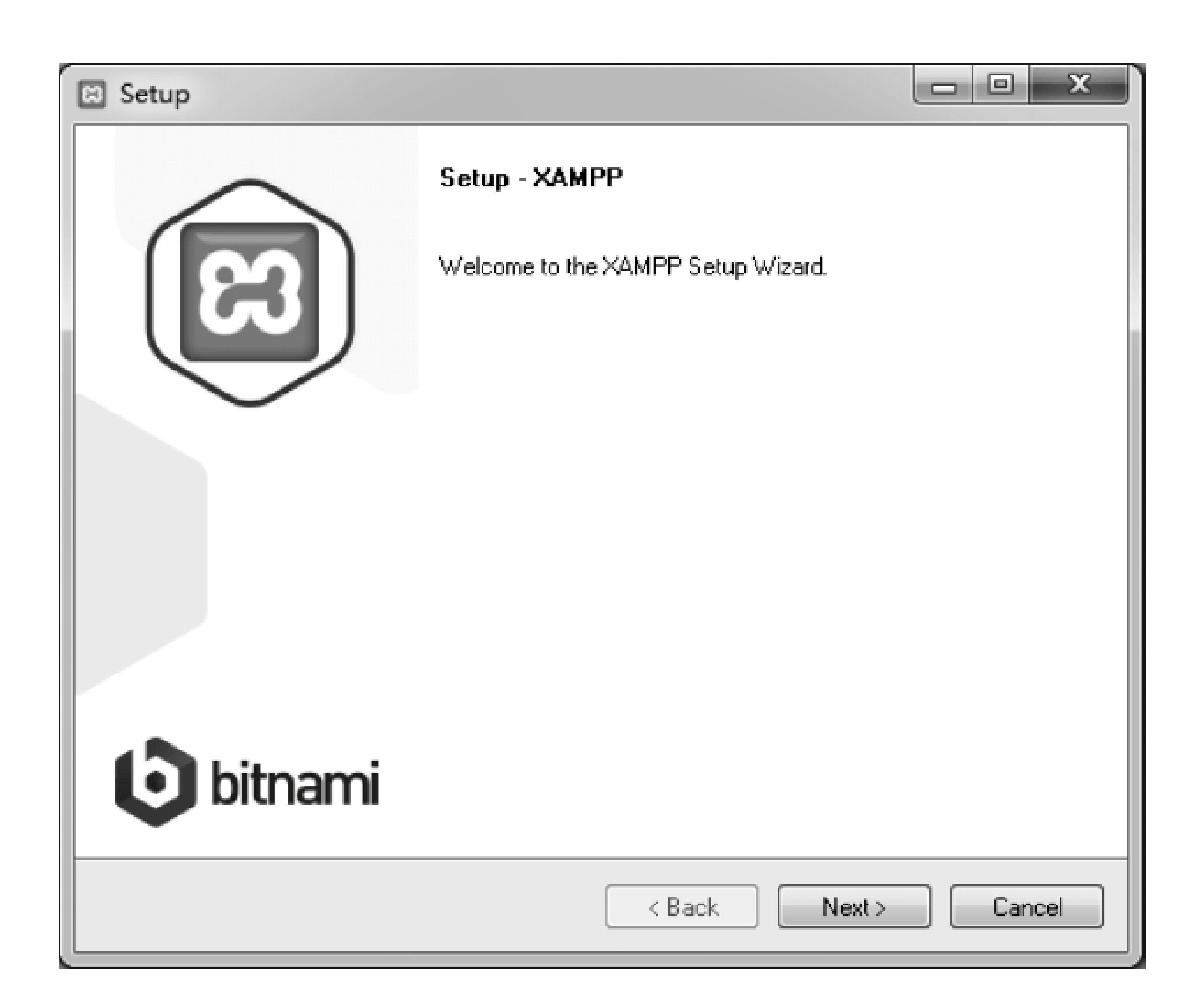


图 1.2　安装主页面对话框

后续对数据库的各种可视化操作。

单击 Next 按钮进入下一步，如图 1.4 所示。

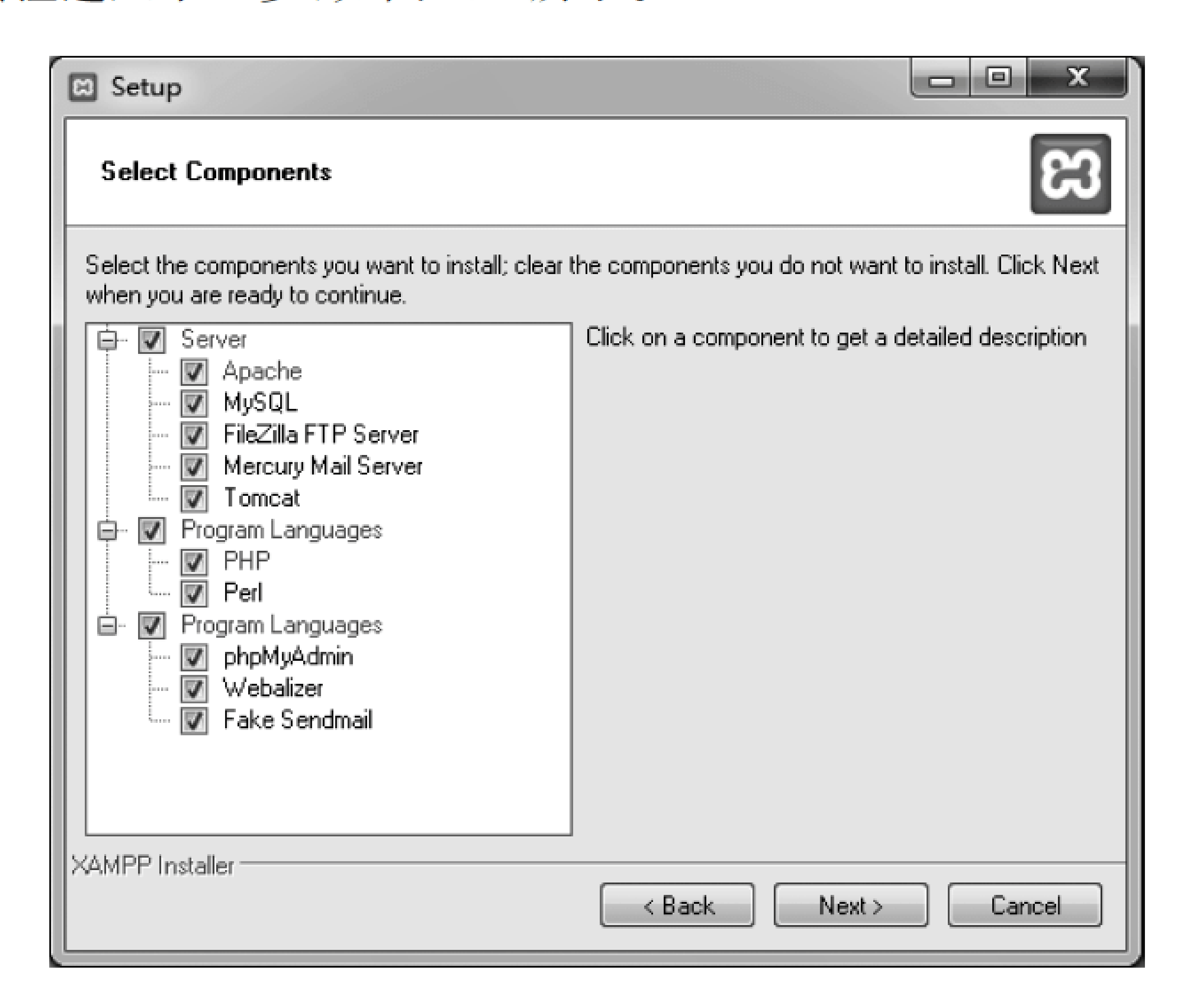


图 1.3　选择组件对话框

(6) 在图 1.4 所示的对话框中选择 XAMPP 软件包的安装路径，单击 Next 按钮开始安装。

(7) 在安装完成对话框中，如果选择了 Do you want to start the Control Panel now? 复选框，单击 Finish 按钮会打开 XAMPP 的控制面板，如图 1.5 所示。

单击图 1.5 所示的 XAMPP 控制面板中各组件对应的 Actions 按钮，可以对相应服务器进行开启与关闭控制。注意开启某项服务时控制面板下部给出的详细信息，如果出现红色错误提示，则需要进行相应的处理。

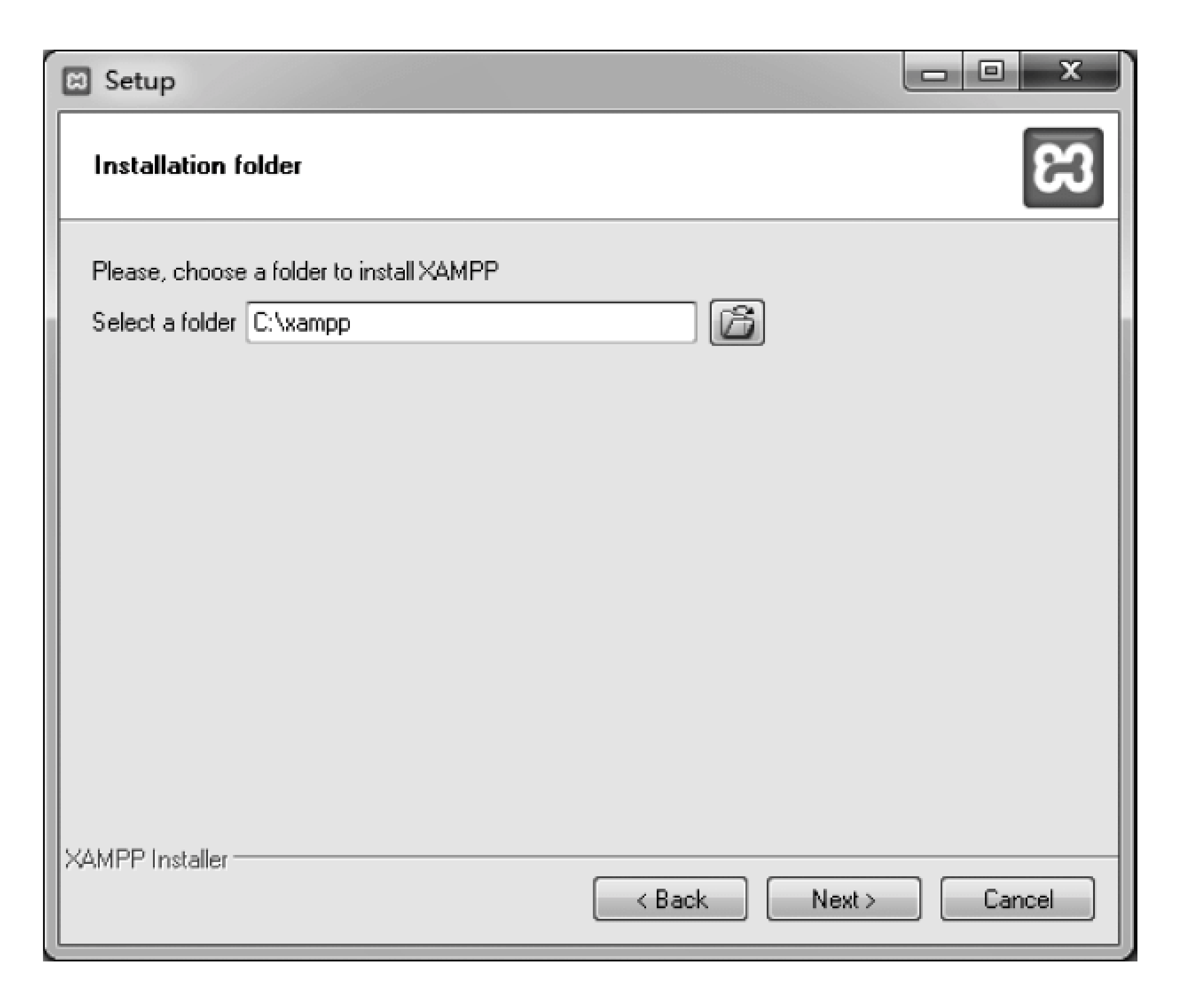


图 1.4 选择安装目录

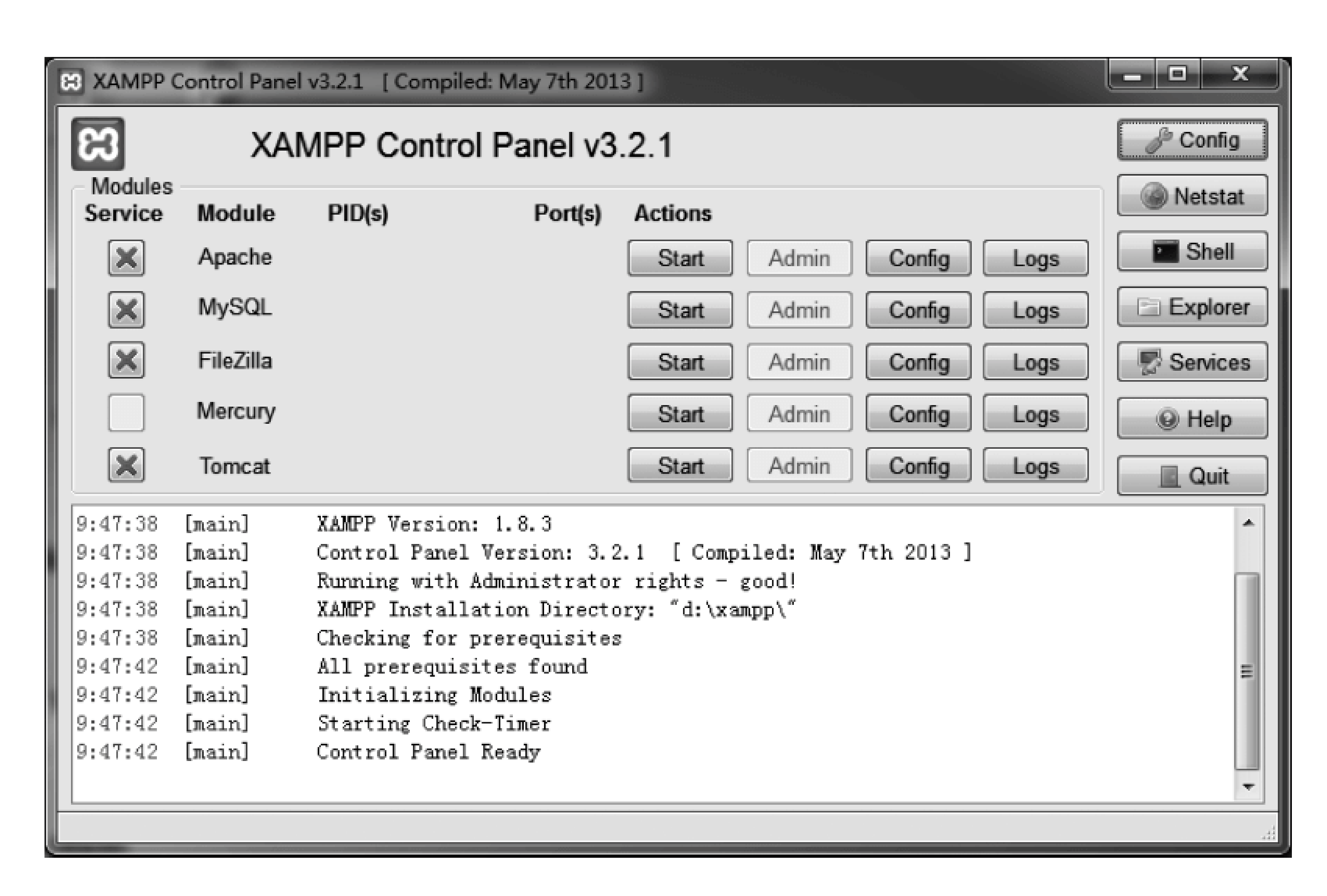


图 1.5 XAMPP 控制面板

（8）安装测试。

软件包安装完成，并成功启动 Apache Web 服务器后，在浏览器地址栏中输入 http://localhost，或者在 XAMPP 控制面板中单击 Apache 对应的 Admin 按钮。如果能看到如图 1.6 所示的页面，则表明 XAMPP 安装成功，可以使用了。

（9）设置 MySQL 数据库服务器密码。

在上述安装过程中，安装程序没有提示输入数据库登录的用户名和密码，默认用户名为 root、密码为空。为了安全起见，最好为数据库设置一个安全密码。

在如图 1.6 所示的页面左侧导航栏中找到 security 链接，单击该链接，打开如图 1.7 所示的页面。然后单击页面中的 http://localhost/security/xamppsecurity.php 链接，为数据库设置登录用户名与密码。

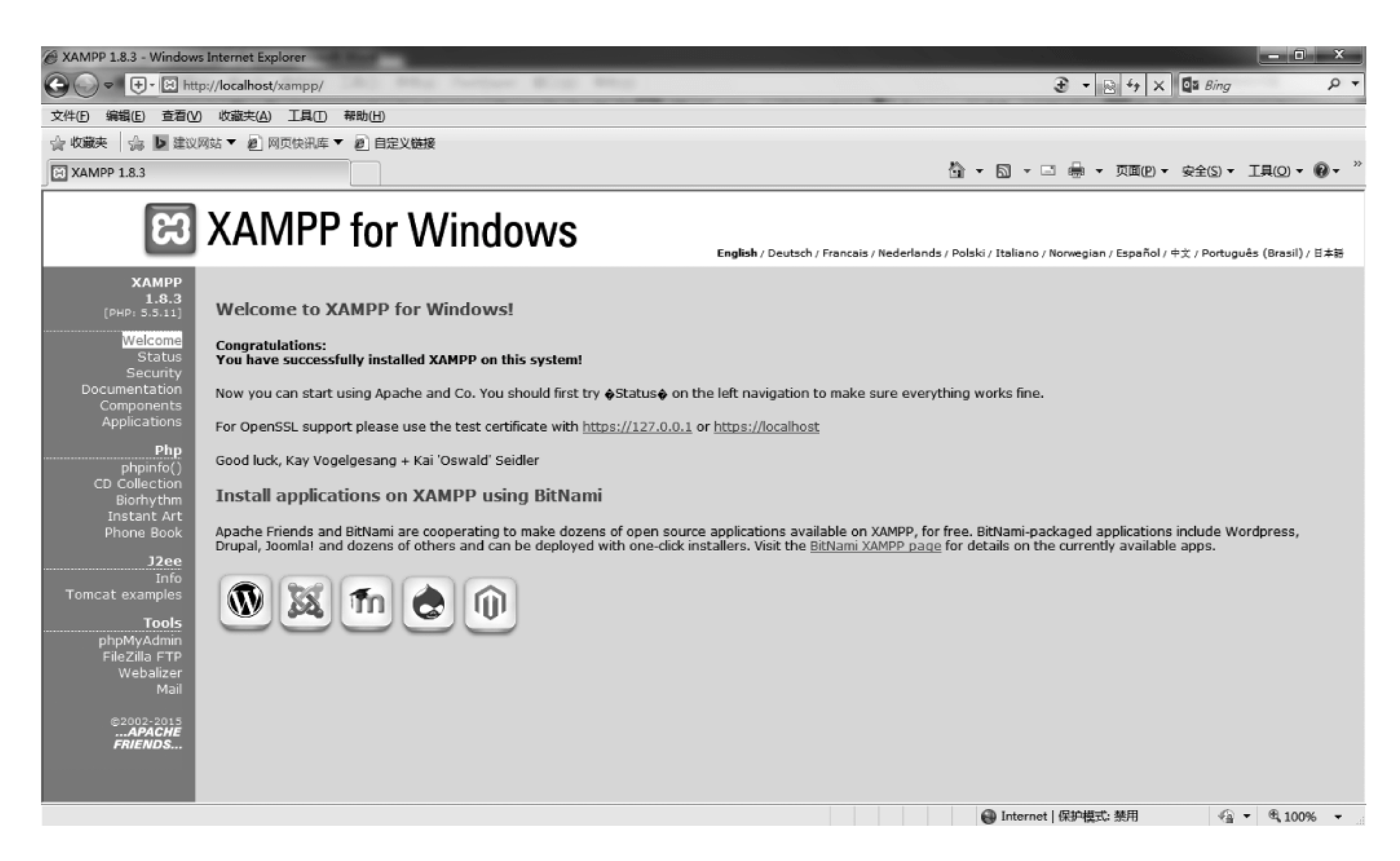


图 1.6　XAMPP 软件包主页面

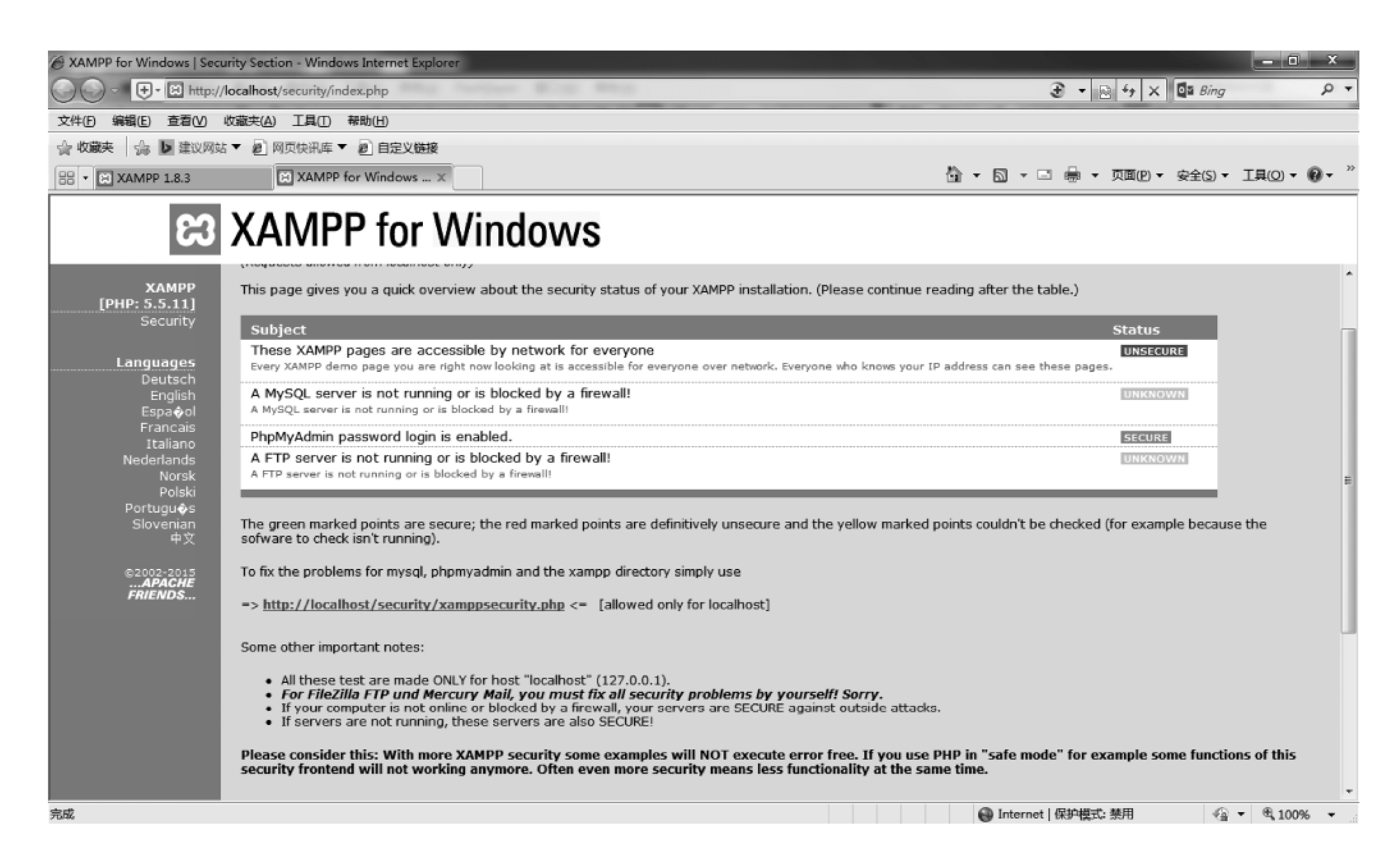


图 1.7　设置数据登录用户名与密码页面

（10）验证 phpMyAdmin 是否安装成功。

在 XAMPP 控制面板中启动 Apache 与 MySQL 服务器，单击 MySQL 对应的 Admin 按钮；或单击图 1.6 所示的页面中、左侧导航栏中的 phpMyAdmin 链接，打开如图 1.8 所示的 phpMyAdmin 数据库管理器主页面。

在页面中输入用户名和密码，即可进入如图 1.9 所示的数据库管理界面。在这里可以

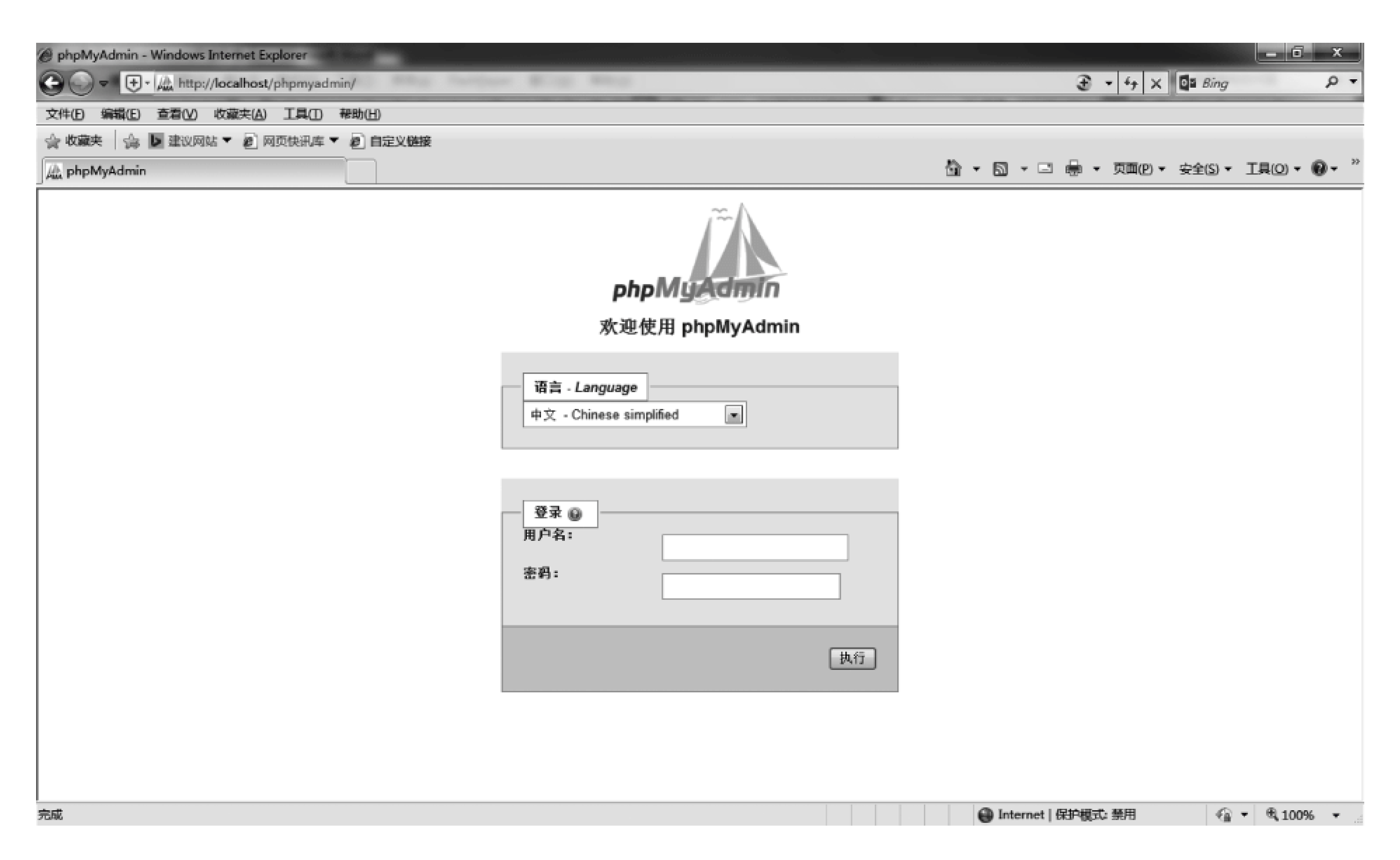


图 1.8　phpMyAdmin 数据库管理器主页面

进行 MySQL 数据库的所有可视化操作，对于不太熟悉数据库命令方式的开发者来说，它是一个得力的好助手。



图 1.9　phpMyAdmin 数据库管理器界面

至此，XAMPP 软件包安装完毕。这时，在计算机系统中就搭建起了一个名为 WAMP 的 PHP 应用开发平台。

图 1.10 所示为 XAMPP 软件包的安装目录，在安装目录下可以看到选择的各种软件的安装文件夹。在根目录下有一个名为 htdocs 的文件夹，它是存放 PHP 脚本的目录，也是项目开发时源文件存入的位置。对于这些目录请大家重点关注，在以后的项目开发过程中对配置文件、数据库文件以及项目文件的各种操作都是在这些文件夹中进行的。



图 1.10　XAMPP 软件包的安装目录

2. AppServ 软件包

AppServ 也是一个目前比较流行的软件工具包，可以为用户快速搭建 PHP 的应用开发环境。它包括 Apache Web 服务器、PHP 应用服务器、MySQL 数据库管理系统和 phpMyAdmin 数据库管理工具 4 个部分。

AppServ 软件包的官方网址为 http://www.appservnetwork.com，目前的最新版本为 2.6。

下面简单介绍 AppServ 软件包的安装步骤：

(1) 打开浏览器，在地址栏中输入上述官方网址，进入 AppServ 官方网站，下载 AppServ 安装软件，这里以 2013-10-25_appserv-win32-2.6.0.1343644756.exe 为例进行示范。

(2) 双击运行下载的 AppServ 安装软件启动安装程序，如图 1.11 所示，然后单击 Next 按钮进入下一步。

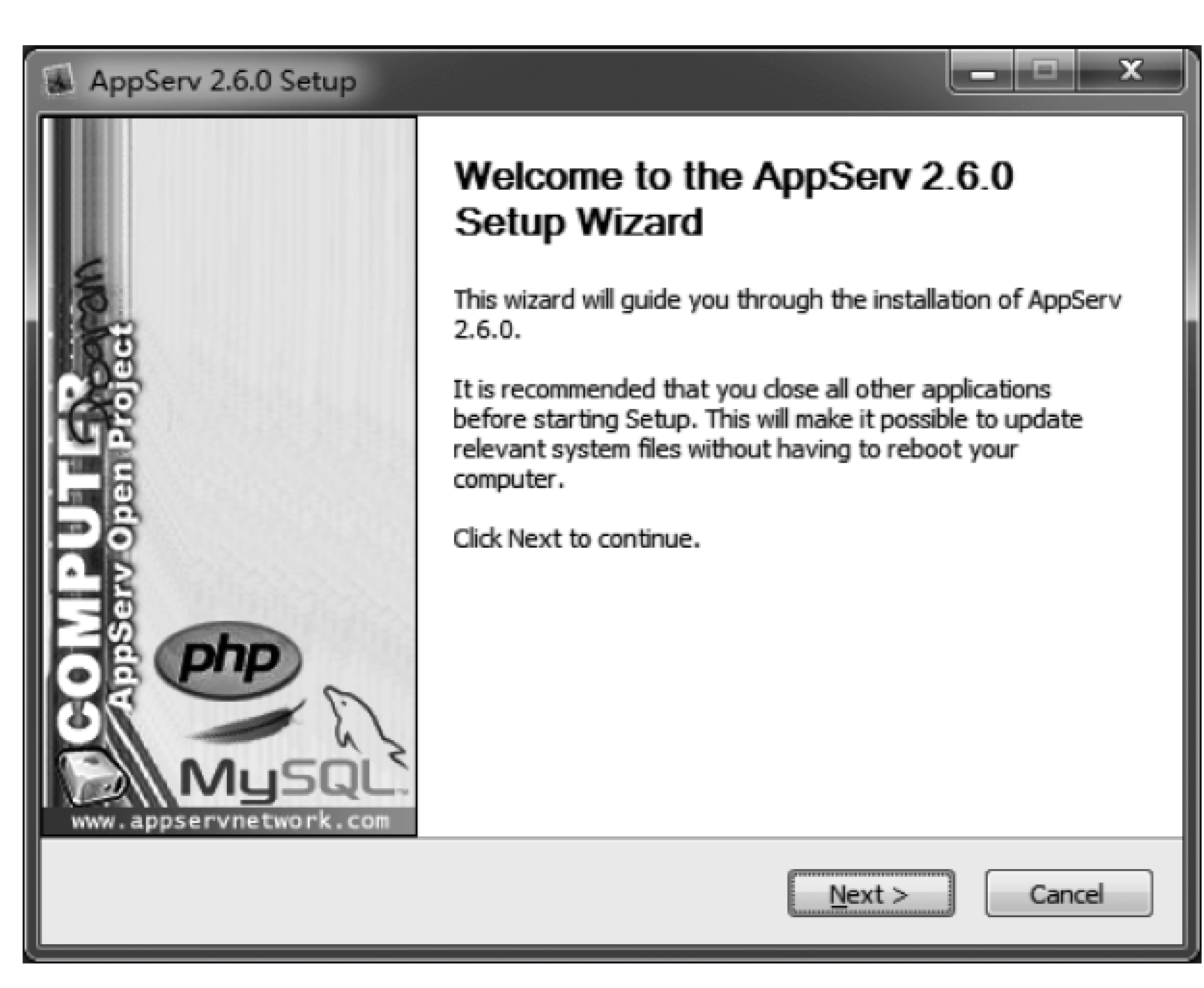


图 1.11　AppServ 软件包的安装对话框

(3) 在是否同意协议的界面中单击 I Agree 按钮,进入安装目录设置界面,如图 1.12 所示。

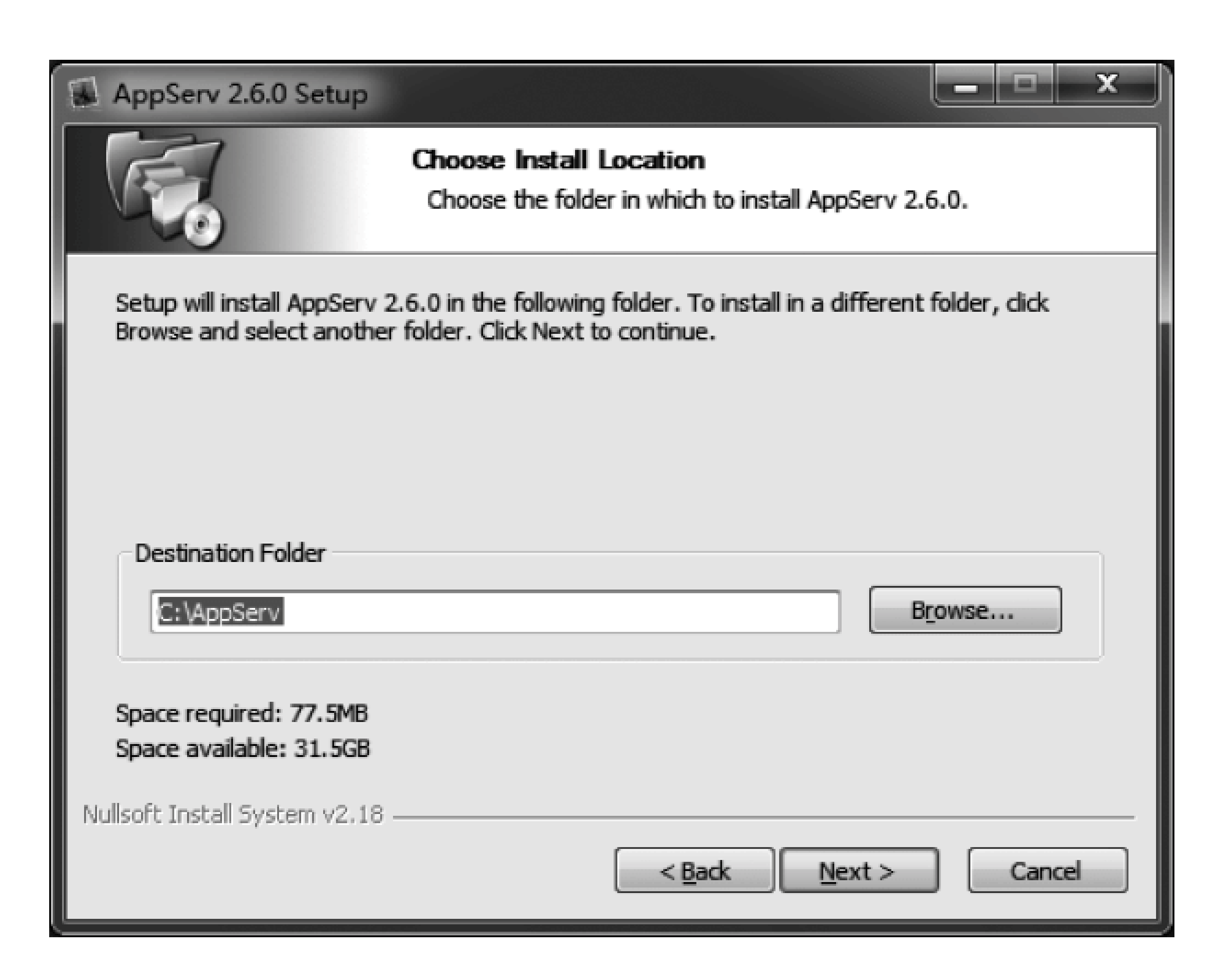


图 1.12　安装目录设置

(4) 选择安装路径后单击 Next 按钮进入组件选择界面,如图 1.13 所示。

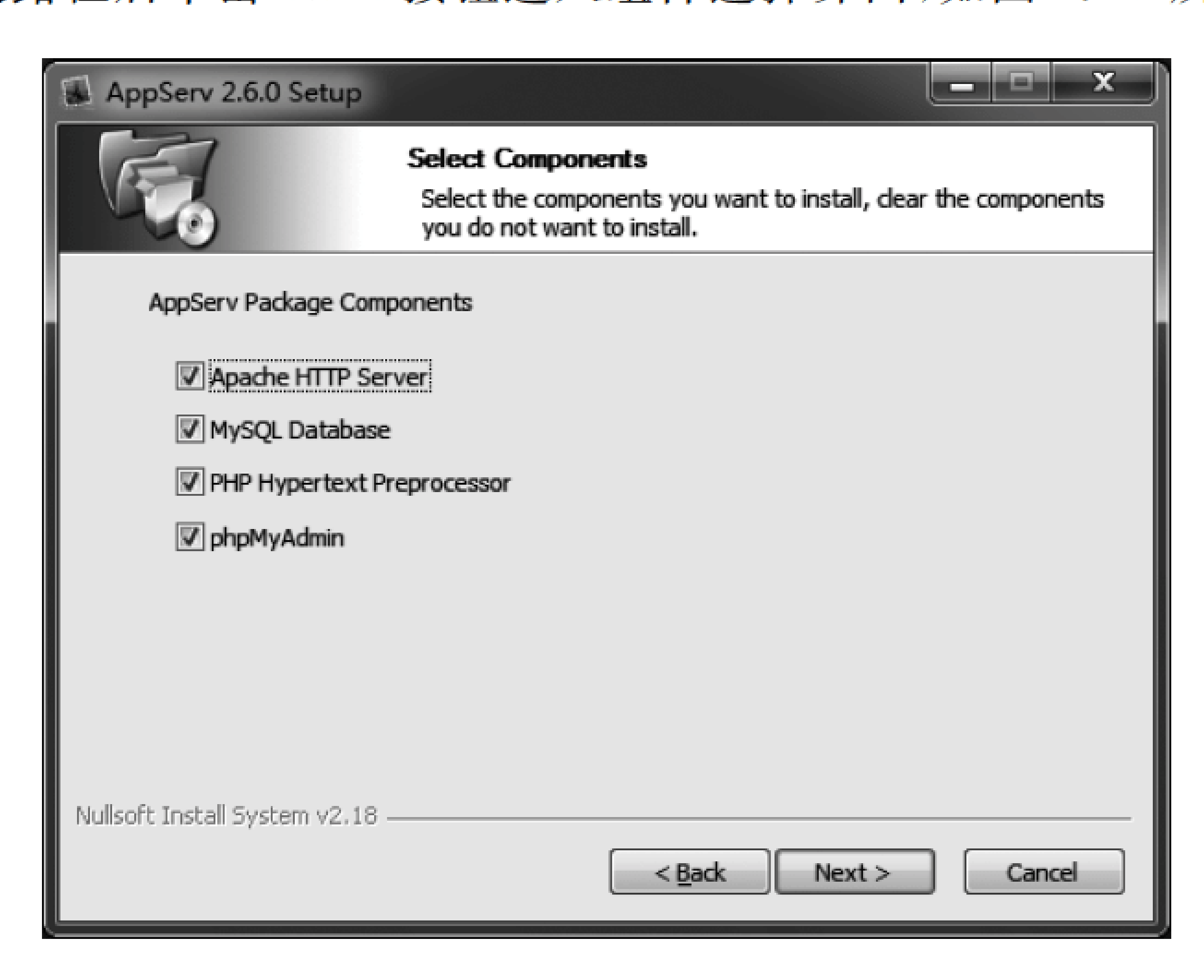


图 1.13　组件选择对话框

(5) 选择界面中的全部组件,单击 Next 按钮,进入服务器信息设置界面,如图 1.14 所示。

(6) 填写 HTTP 服务器的地址,即 127.0.0.1,邮箱地址可以随便写,保持端口号为 80,然后单击 Next 按钮,进入 MySQL 数据库设置界面,如图 1.15 所示。

(7) 配置 MySQL 服务器的 root 用户密码,保持字符集为 UTF-8,选择 Enable InnoDB 复选框,单击 Install 按钮开始安装。

(8) 软件包安装完成,选择并启动 Apache 和 MySQL 服务器后,在浏览器中输入 http://localhost,如果出现如图 1.16 所示的页面,则表示 AppServ 软件包安装成功。

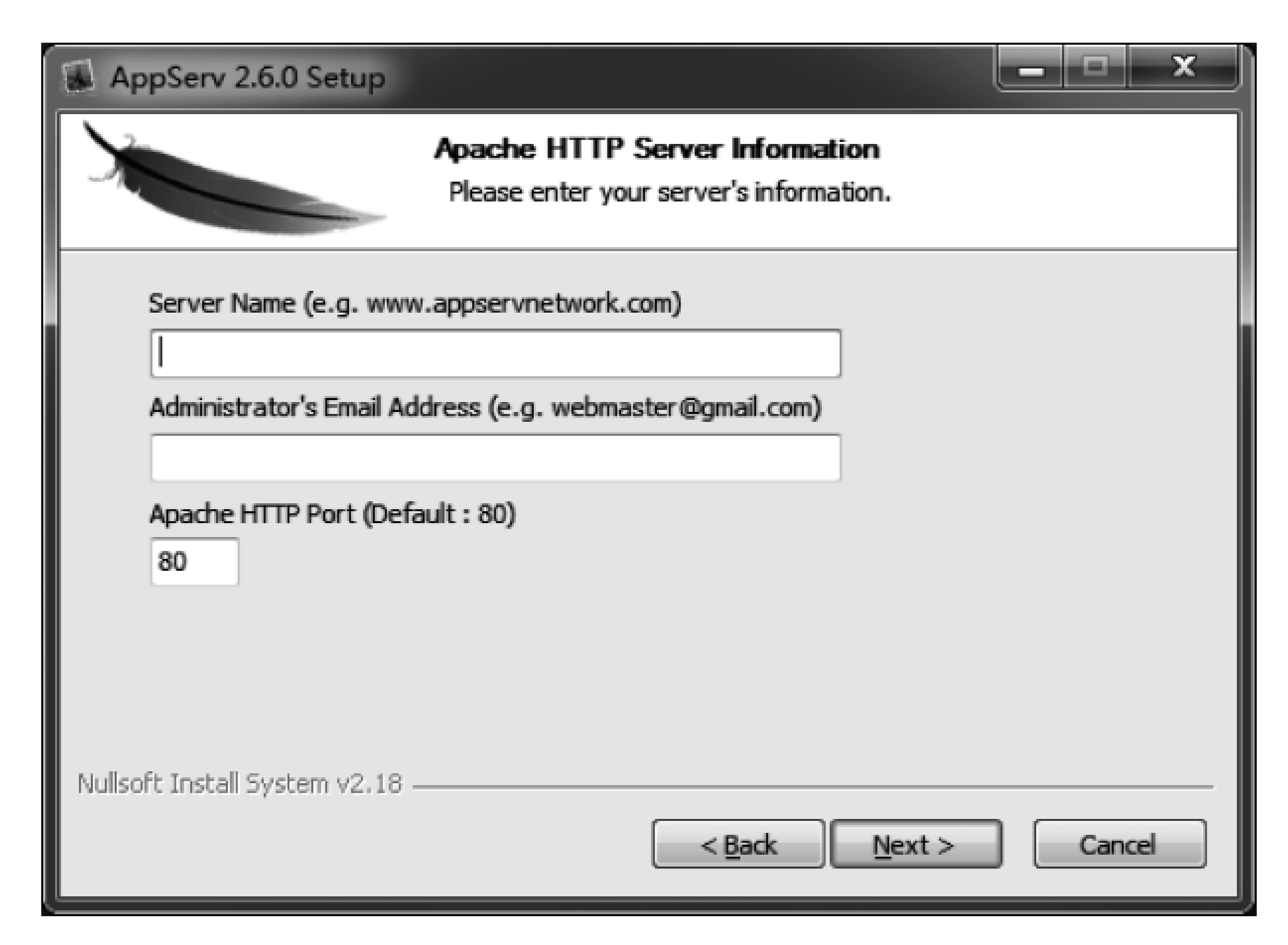


图 1.14　服务器信息设置

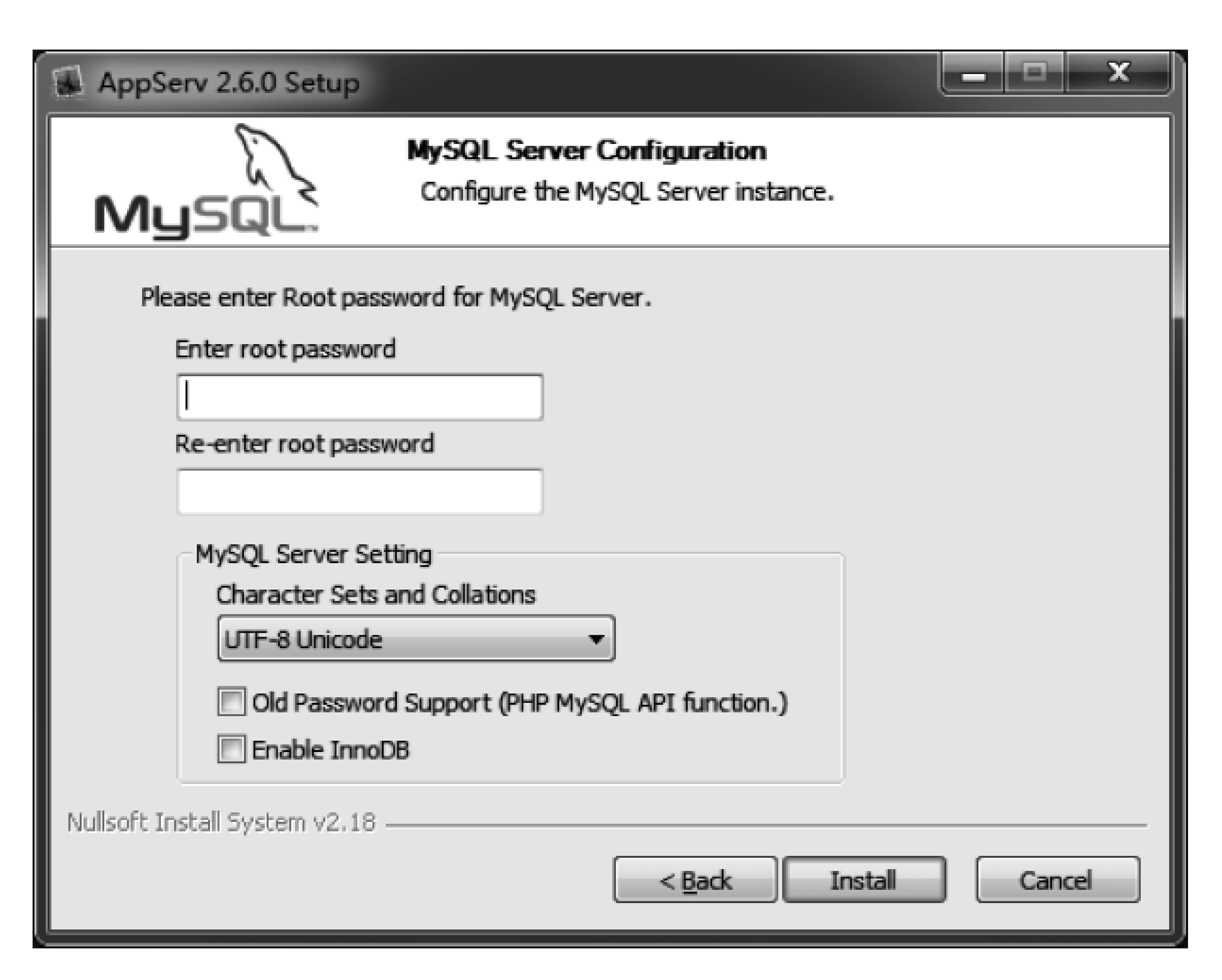


图 1.15　MySQL 数据库设置界面

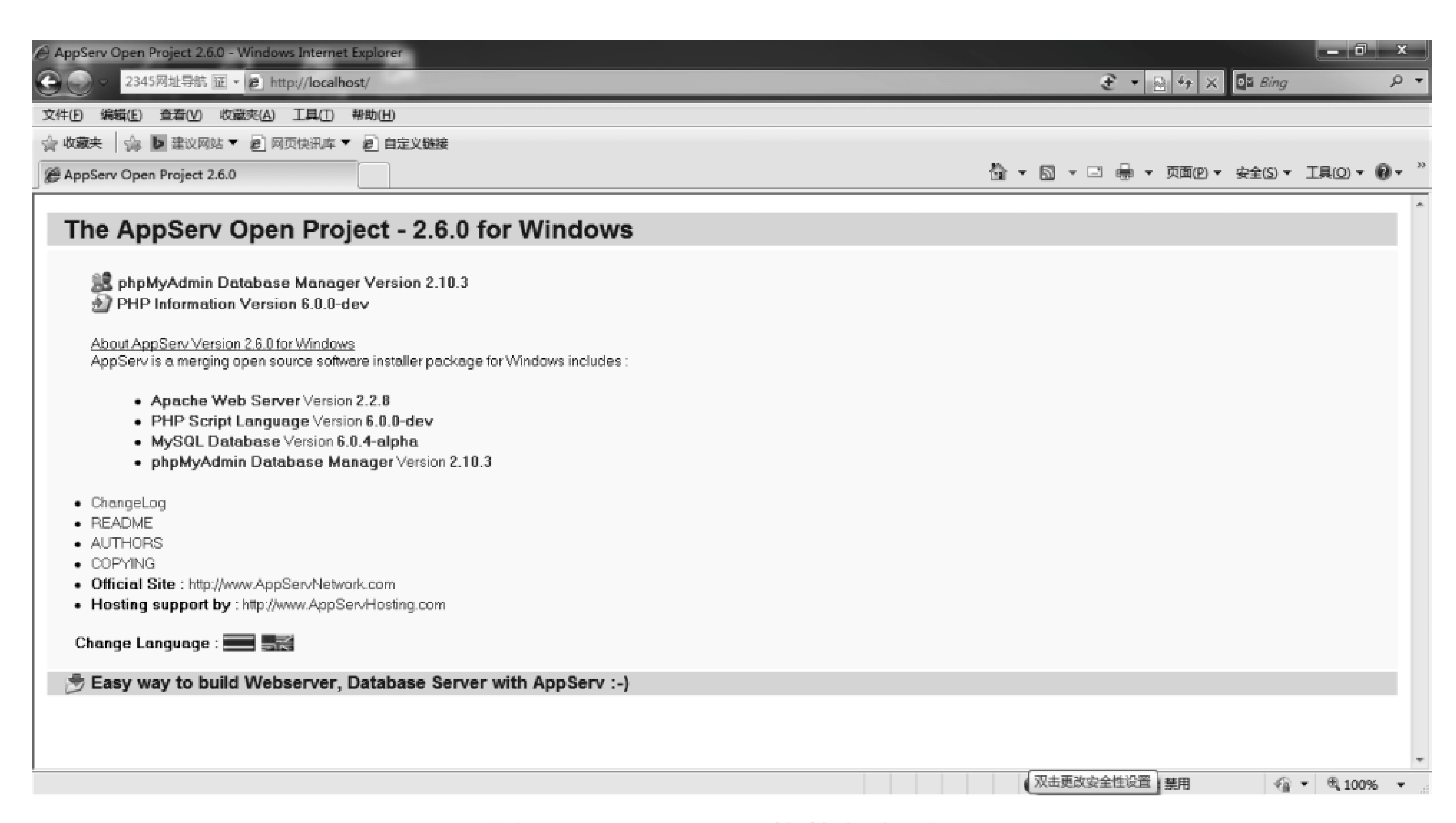


图 1.16　AppServ 软件包主页面

打开 AppServ 软件包的安装目录，可以看到我们所选择的 4 个组件的文件夹。安装目录中的 www 目录是项目脚本的存放目录。本书案例项目采用 XAMPP 开发环境，关于 AppServ 集成开发环境的使用与配置，请大家查询其他的技术资料。

1.2.2 设置虚拟主机

XAMPP 集成软件包安装成功后，就在计算机中搭建了 WAMP 的开发平台，这样就可以进行一般的 PHP 应用开发了，但要开发基于 Zend Framework 框架的应用程序，还必须设置一台虚拟主机来模仿真实的服务器环境。

为什么要设置虚拟主机才能运行 Zend Framework 项目呢？这是由 Zend Framework 框架的路由规则决定的。关于这一点将在下一章详细介绍，这里大家先掌握配置虚拟主机的方法。

假设开发的项目名称为 wmProject，存放在 XAMPP\htdocs 目录中，即 Web 服务器的脚本目录中。对于一般的 PHP 应用程序，这个目录就是项目的根目录，首页文件 index.php 就应该存放在这里。但由于 Zend Framework 是一个框架，具有特定的目录结构，它的首页文件 index.php 是存放在项目文件夹的 public 目录下的，所以虚拟主机要直接指向 XAMPP\htdocs\wmProject\public 目录。

1. 添加本地系统 DNS

成功安装 XAMPP 后的计算机系统就已经是一台本地服务器了，通过在浏览器的地址栏中输入 http://localhost 可以访问存放在 XAMPP\htdocs 目录中的应用程序，例如网站、办公系统等，也就是说，通过 localhost 这个域名就可以访问服务器的根目录 htdocs。通过修改系统配置文件可以为本机设置其他域名，例如 wmoams.com。

(1) 打开系统目录 C:\Windows\System32\drivers\etc，用文本编辑工具打开配置文件.hosts，其内容如下：

```
# Copyright (c) 1993 - 2009 Microsoft Corp.
#
…
# For example:
#
#       102.54.94.97     rhino.acme.com          # source server
#        38.25.63.10     x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
    127.0.0.1        localhost
#       ::1              localhost
#       127.0.0.1 activate.adobe.com
```

从代码中可以看到，localhost 对应本机 IP 地址 127.0.0.1，所以在浏览器的地址栏中输入 http://localhost 或 http://127.0.0.1 都可以访问服务器的 htdocs 根目录。

(2) 在系统配置文件.hosts 的末尾添加如下代码，为本机添加新域名 wmoams.com。

```
127.0.0.1    wmoams.com
```

保存后关闭.hosts 配置文件，在 XAMPP 控制面板中重启 Apache 服务器。在浏览器

的地址栏中输入 http://wmoams.com，如果能打开如图 1.6 所示的 XAMPP 集成开发环境的主页面，则说明域名配置成功。

2. 开启 Apache 的 Rewrite 功能

打开如图 1.5 所示的 XAMPP 控制面板，单击 Apache 对应的 Config 按钮，打开配置文件 httpd.conf。该配置文件位于 D:\xampp\apache\conf 目录下，当然也可以在文件夹下直接打开文件。在文件中找到如下代码：

```
#LoadModule reqtimeout_module modules/mod_reqtimeout.so
LoadModule rewrite_module modules/mod_rewrite.so
#LoadModule sed_module modules/mod_sed.so
```

确保 LoadModule rewrite_module 前的#注释符号是去除的，这样就开启了 Apache 服务器的 Rewrite 功能，可以对来自 HTTP 的请求进行重定向处理。

这里 XAMPP 已经进行了配置。不同的 XAMPP 版本，初始配置不尽相同，当然，不同的 Apache 的安装方式更是有不同的配置方法，请大家特别注意这一点。

3. 添加虚拟主机

(1) 打开配置文件 D:\xampp\apache\conf\httpd.conf，找到如下代码：

```
#Virtual hosts
Include conf/extra/httpd-vhosts.conf
```

确保上述代码中 Include 前面的#注释符号是去除的，这句代码的意思是启用 Apache 服务器的虚拟主机配置文件 httpd-vhosts.conf。

(2) 打开 Apache 服务器配置文件 D:\xampp\apache\conf\extra\httpd-vhosts.cof，找到如下代码：

```
# Use name-based virtual hosting.
#
NameVirtualHost *:80
#
```

确保代码中的 NameVirtualHost 前面的#注释符号是去除的。代码前的注释，表明将采用重新定义名称的方式设置虚拟主机。除了重命名的方式以外，用户还可以采用重新设置端口的方式进行虚拟主机的设置。

(3) 在 Apache 服务器配置文件 httpd-vhosts.cof 后添加如下代码：

```
<VirtualHost *:80>
  DocumentRoot "D:/xampp/htdocs"
  ServerName localhost
</VirtualHost>

<VirtualHost *:80>
    DocumentRoot "D:/xampp/htdocs/wmProject/public"
    ServerName wmoams.com
</VirtualHost>
```

这段代码分为两个单元，下面的单元是新设置的名为 wmoams.com 的虚拟主机，根目

录是 D:\xampp\htdocs\wmProject\public，也就是项目 wmProject 的根目录；上面的单元用于确保原来的 localhost 域名不会失效。

(4) 测试虚拟主机是否配置成功。

打开项目的 D:\xampp\htdocs\wmProject\public 文件夹，用记事本新建一个 index.php 文件，代码如下：

```
<?php   echo "http://www.wmstudio.net.cn ------ Zend Framework project! ";
```

重新启动 Apache 服务器，在浏览器地址栏中输入 http://wmoams.com，如果能看到如图 1.17 所示的页面，说明虚拟主机设置成功。

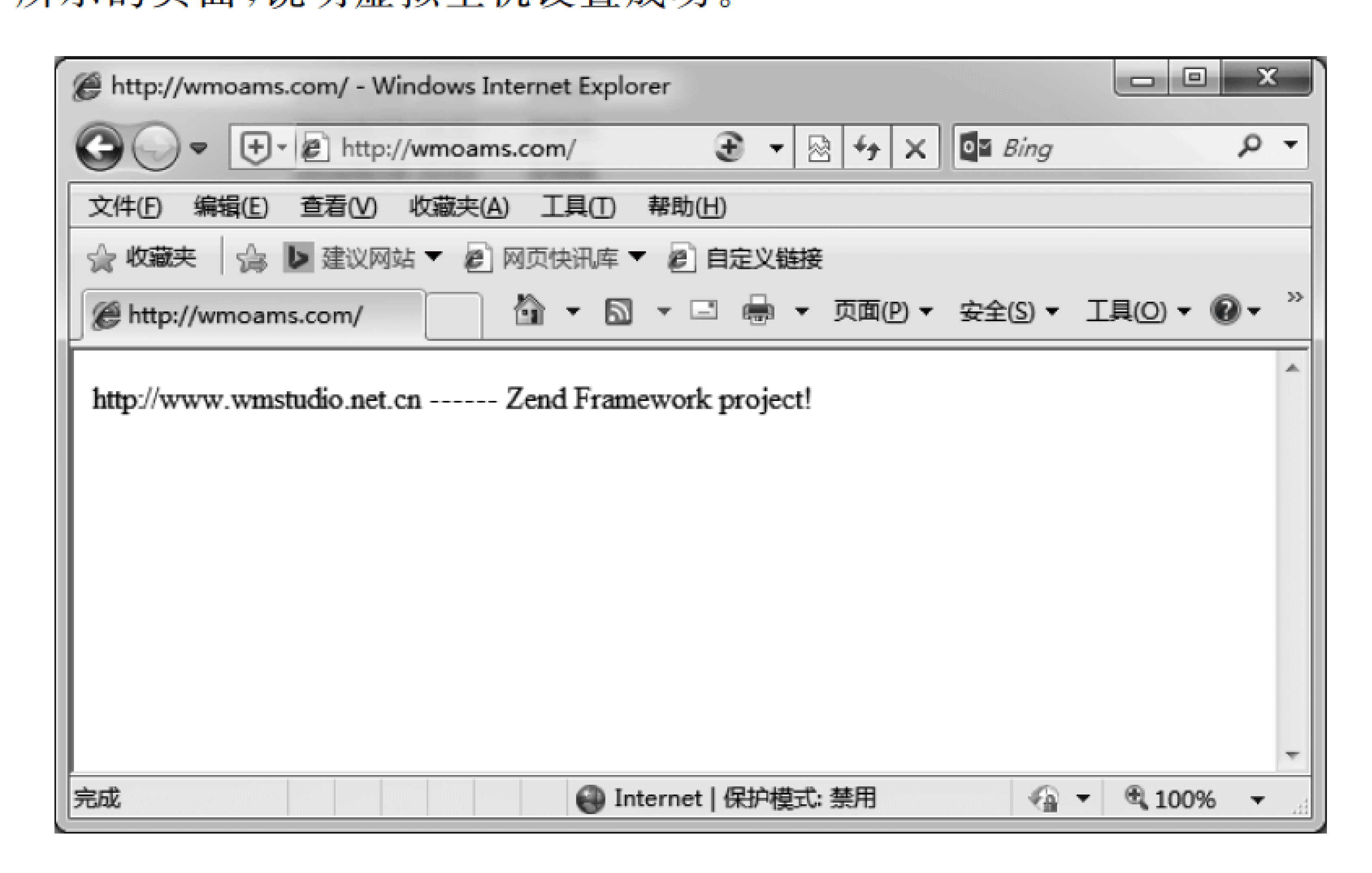


图 1.17　虚拟主机测试页面

1.2.3　开发环境的配置

上面通过 XAMPP 集成软件包搭建起了经典的 WAMP 开发环境，并且发现许多配置已经设置完成。这表明在系统安装过程中，安装程序会自动进行一些基础配置，这也正是我们选择集成软件包创建开发环境的原因。

下面以作者计算机中的软件安装目录为例，简单介绍 WAMP 开发环境配置过程中涉及的一些配置文件。

1. Windows 系统配置文件

Windows 配置文件有很多，这里我们用到的只有 hosts 文件，它位于 C:\Windows\System32\drivers\etc 目录下，主要用来设置本机服务器的新域名。

2. PHP 配置文件

1) 使用 PHP 扩展库

PHP 配置文件 php.ini 位于 PHP 安装根目录下，在这里大家可以发现多个 INT 配置文件，例如 php.ini-development、php.ini-production 等，它们是 PHP 开发者为用户准备的配置文件模板，大家应根据项目开发的不同阶段选用不同的配置模板文件。

打开目录中的 php.ini 配置文件，可以发现文件里有各种各样的配置选项，要全部弄清

每项配置的含义不是一件容易的事情，请大家特别关注文件中如下形式的代码：

```
⋮
extension = php_bz2.dll
extension = php_curl.dll
extension = php_mbstring.dll
extension = php_exif.dll
;extension = php_fileinfo.dll
extension = php_gd2.dll
extension = php_gettext.dll
;extension = php_gmp.dll
;extension = php_intl.dll
;extension = php_imap.dll
;extension = php_interbase.dll
;extension = php_ldap.dll
;extension = php_mssql.dll
;extension = php_mbstring.dll
⋮
```

以上是启用 PHP 扩展库的代码，去掉 extension 前面的“;”注释符号，表示加载了相应的 PHP 扩展库。PHP 库文件存放在 PHP 根目录的 pear 文件夹下，动态链接库的 *.dll 文件存放在 PHP 根目录下。在上面的代码中去掉了注释符号的语句，它是在安装 XAMPP 过程中系统自动配置的。

2）默认时区设置

打开 PHP.ini 配置文件，找到如下代码，将默认时区设置为 PRC。

```
[Date]
; Defines the default timezone used by the date functions
; http://php.net/date.timezone
;date.timezone = Europe/Berlin
date.timezone = PRC
```

3）默认字符集设置

打开 PHP.ini 配置文件，找到如下代码，将默认字符集设置为 utf-8。

```
; PHP's default character set is set to empty.
; http://php.net/default - charset
default_charset = "utf - 8"
```

3. Apache 服务器配置文件

Apache 服务器配置文件有标准配置与扩展配置两种类型。

1）Apache 标准配置文件 httpd.conf

该文件位于 D:\xampp\apache\conf 目录下，主要完成一些服务器的基础配置工作。

2）Apache 扩展配置文件

在 Apache 服务器的 D:\xampp\apache\conf\extr 目录下有很多配置文件，这些都是 Apache 的扩展配置文件，例如上面用到的 httpd-vhosts.conf 文件。启用这些配置文件需要打开 httpd.conf 中相应的 include 语句，在上面设置虚拟主机的过程中，我们就打开了 httpd.conf 中的语句：

```
Include conf/extra/httpd-vhosts.conf
```

4. MySQL 数据库配置文件

MySQL 数据库配置文件 my-default.ini 位于 D:\xampp\mysql 目录下，主要用于完成 MySQL 数据库管理系统的配置工作。

1.2.4 Zend Framework 的安装

上面已经成功搭建并配置了 WAMP 的 PHP 应用程序开发环境，如果要运行 Zend Framework 项目，还需要安装 Zend Framework。

Zend Framework 的官方下载地址为 http://www.zend.com，下载后的 Zend Framework 是一个压缩包。将压缩包解压，可以看到里面有各种各样的文件夹及文件，但并没有常见的可执行安装程序。

前面介绍了 Zend Framework 的结构及常用组件的功能，这些组件其实就是一些类库。所以，所谓 Zend Framework 安装实质上是让 PHP 能访问 Zend Framework 的库文件。

1）查看 PHP 的 include_path 路径

打开 XAMPP 的控制面板，启动 Apache 服务器，在浏览器的地址栏中输入 http://localhost，打开如图 1.6 所示的页面。单击页面左侧导航栏中的 phpinfo()链接，打开 phpinfo 信息页面，找到 include_path 显示项，如图 1.18 所示。

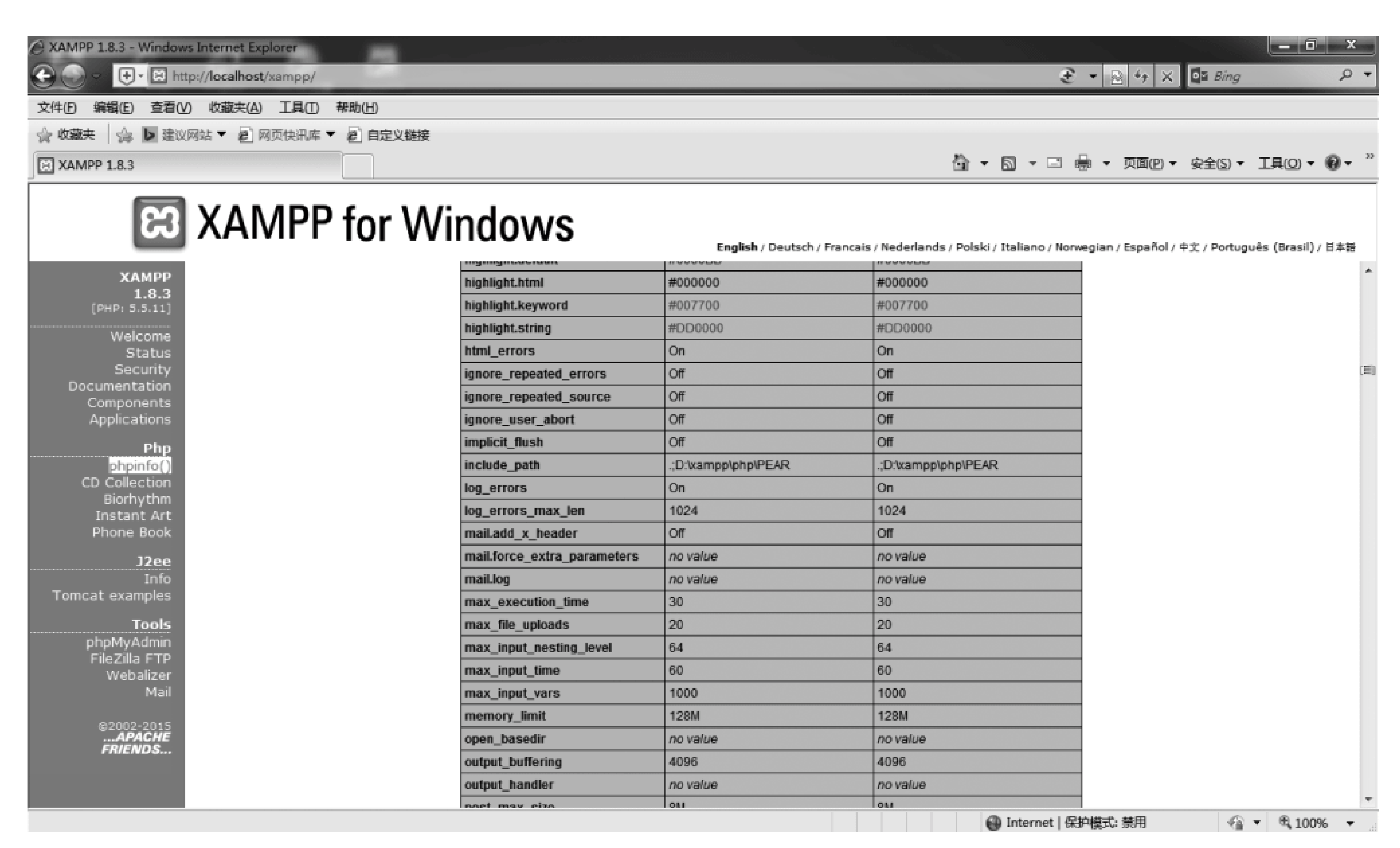


图 1.18　phpinfo 信息页面

从图 1.18 中可以看出，PHP 的 include_path 路径为 D:\xampp\php\PEAR。将 Zend Framework 解压后的 library\Zend 文件夹复制到该路径下，Zend Framework 就安装完成了。

2）测试 Zend Framework

如果要测试 Zend Framework 能否正常工作，最简单的方法就是直接引用（不带路径）

Zend Framework 组件,如果程序不报错就说明 Zend Framework 安装成功。

在上面创建的项目 wmProject 的 public 目录下新建文件 date.php 并添加如下代码:

```
<?php
  require_once('Zend/Date.php');
  $date = new Zend_Date();
  echo Date('Y-m-d', $date->get());
?>
```

在浏览器的地址栏中输入 http://wmoams.com/date.php,如果能看到如图 1.19 所示的页面效果,则说明 Zend Framework 成功安装。

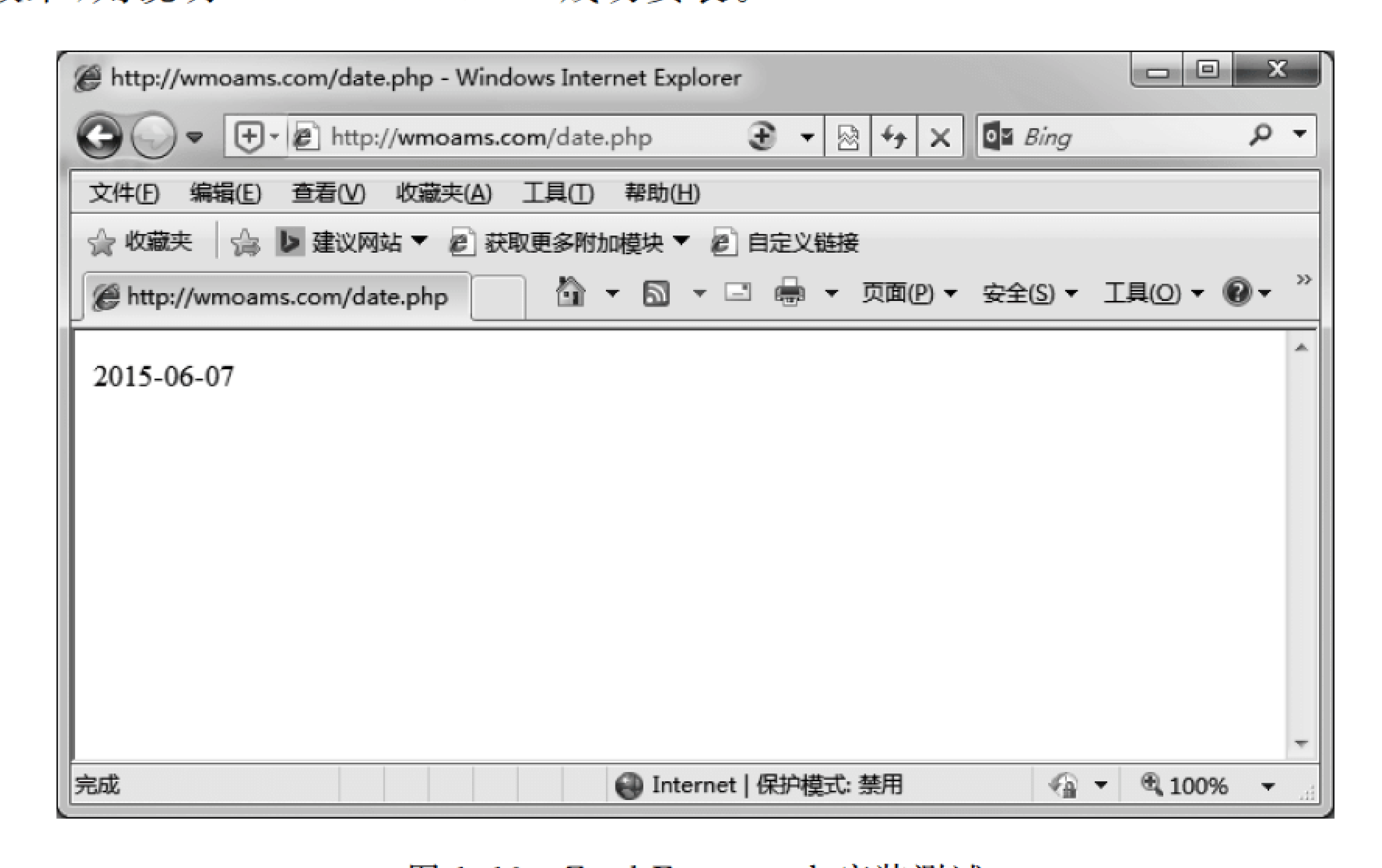


图 1.19 Zend Framework 安装测试

1.3 开发工具与技术文档

上面创建的 WAMP 开发环境只是 PHP 应用程序的调试与运行环境,如果要进行项目的开发,还需要一些 PHP 脚本查看与编辑工具。另外,在 PHP 项目的开发过程中需要编写一些 HTML 网页文件、CSS 样式文件以及 JavaScript 脚本文件,同样需要用相应的开发工具来完成。下面介绍两款常用的 PHP 应用开发工具。

1.3.1 Zend Studio 集成开发环境

Zend Studio 是 Zend Technologies 公司开发的 PHP 语言集成开发环境,具备功能强大的专业编辑工具和调试工具,支持 PHP 语法高亮显示,并有智能语法提示功能,可以大大提高开发效率。另外,它还支持语法自动填充、语法自动缩排、代码复制和书签等功能。

Zend Studio 集成开发环境内置的调试工具功能强大,支持本地和远程两种调试模式,并支持多种高级调试功能,例如跟踪变量、单步运行、断点、堆栈信息、函数调用、查看实时输出等。此外,它对中文也有非常好的支持。

Zend Studio 包含了 PHP 应用开发所有必需的部件,通过一整套编辑、调试、分析、优化

和数据库工具极大地缩短了开发周期，并简化了复杂的应用方案。毋庸置疑，它已经是目前最强大的 PHP 集成开发环境，当然也是 Zend Framework 项目开发的首选。

Zend Studio 集成开发环境的官方下载地址为 http://www.zend.com，能提供多种版本的下载。Zend Studio IDE 是一款收费软件，试用版本有 30 天的试用期。本书作者使用的是目前比较新的 10.6 版。

Zend Studio 的 Windows 版本是一个以.msi 为扩展名的程序包，双击打开即可安装。安装成功后打开软件，界面如图 1.20 所示。下面简单介绍 Zend Studio 的功能及使用中需要注意的一些问题。

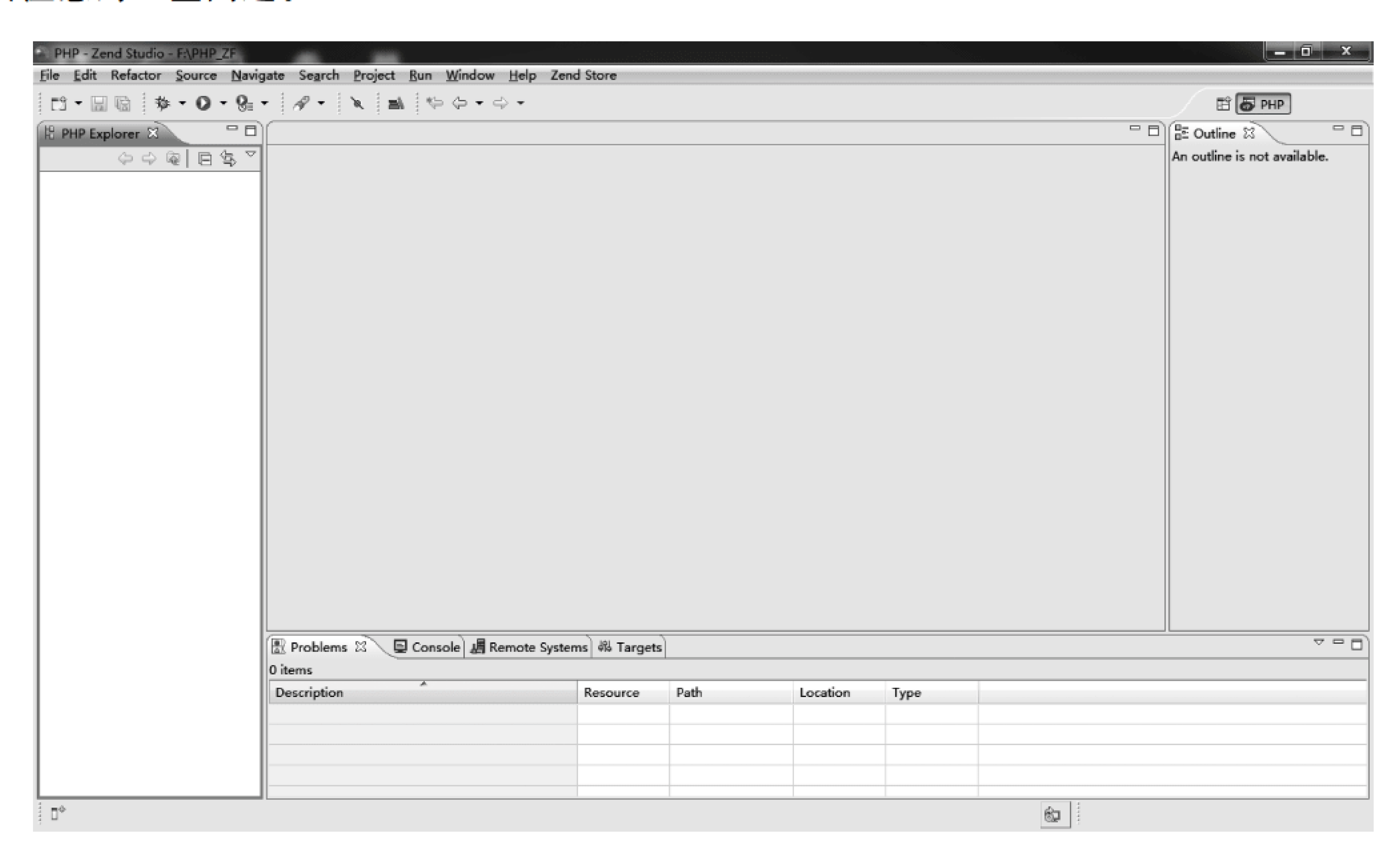


图 1.20　Zend Studio 集成开发环境界面

1. 工作台的切换

Zend Studio IDE 是一个功能强大的开发环境，它不仅可以编辑、调试 PHP 代码，还可以编辑、调试其他类型的文件。如图 1.30 所示的界面为 PHP 项目工作界面，这也是它的默认工作台界面。

选择菜单栏中的 Window|Open Perspective 菜单项，或者单击如图 1.21 所示的界面右上角的工作台切换图标，都可以对工作台进行切换。

2. 工作区的切换

与其他集成开发环境一样，例如 Visual Studio、Eclipse、myEclipse 等，Zend Studio 对项目的管理也是以工作区的形式进行的。如图 1.20 所示，界面左侧区域即 Zend Studio 的项目工作区，在这里显示当前工作区中的所有项目。有时我们会将不同类型的应用项目放在不同的工作区中，这样就需要进行工作区的切换。

Zend Studio 工作区的选择有两种方法，一是在启动 Zend Studio 时对工作区进行选择，另外就是在 Zend Studio 打开以后进行切换。Zend Studio 启动后工作区的切换是通过选择 File|Switch Workspace 菜单项实现的。

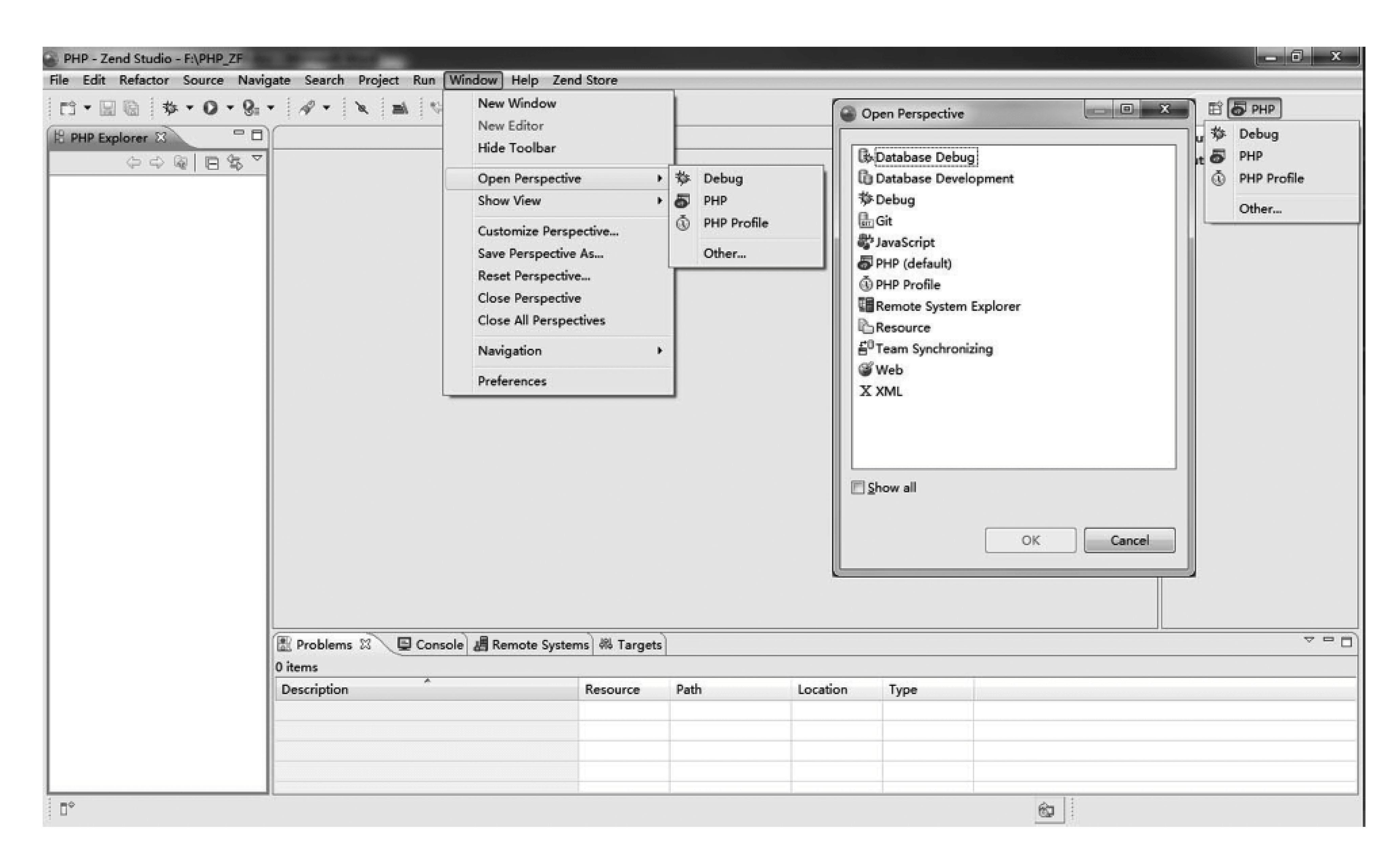


图 1.21　Zend Studio 工作台的切换

3. 工作区属性设置

工作区属性设置后，对本工作区中的每一个项目都是有效的。选择 Window|Perferences 菜单项，打开属性设置对话框，如图 1.22 所示。

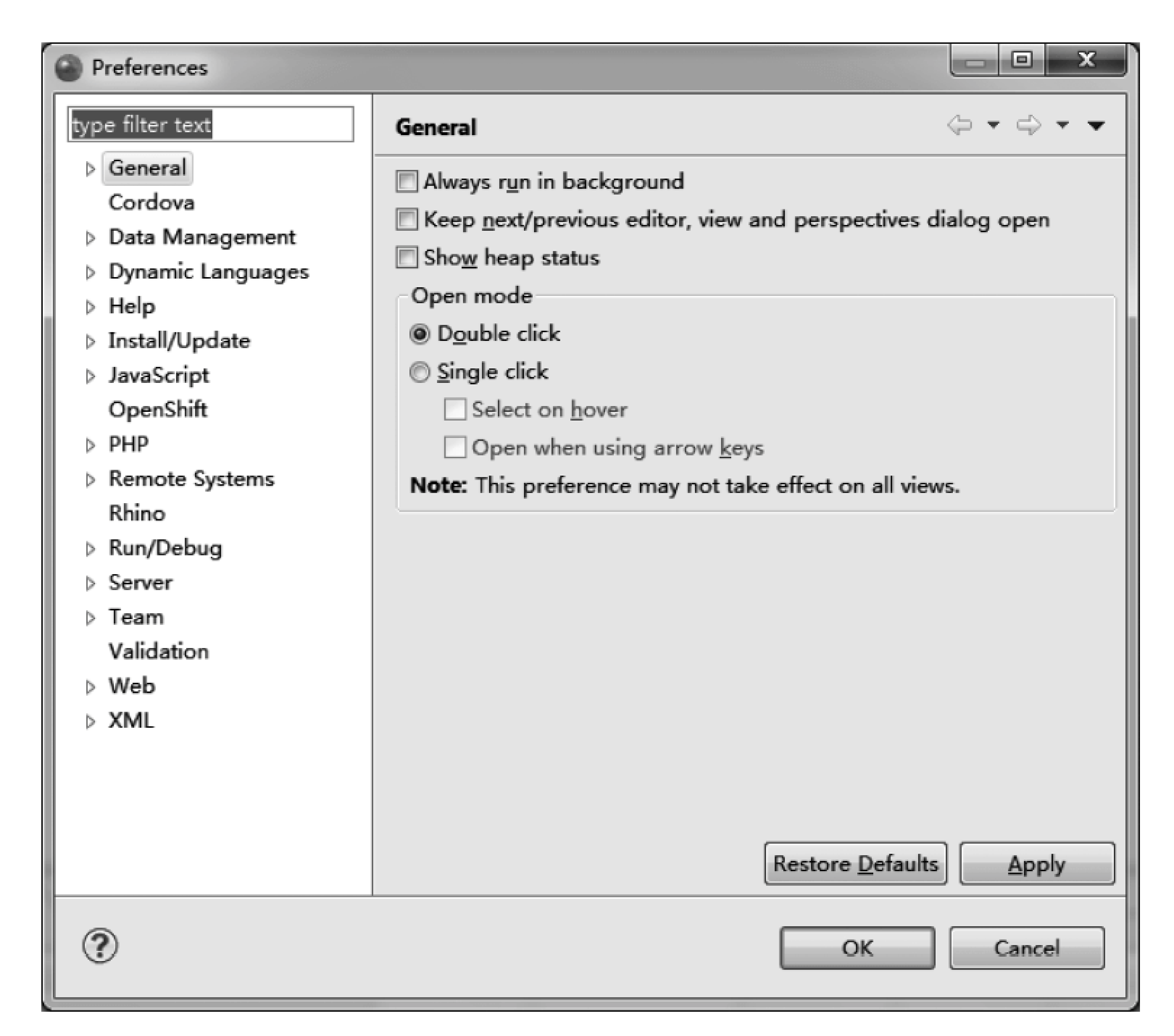


图 1.22　Zend Studio 工作区属性设置

4. 项目属性设置

在工作区中选择某个项目，选择 Project|Perferences 菜单项，打开项目属性设置对话



框，如图 1.23 所示。注意，在这里设置的属性只对本项目有效。

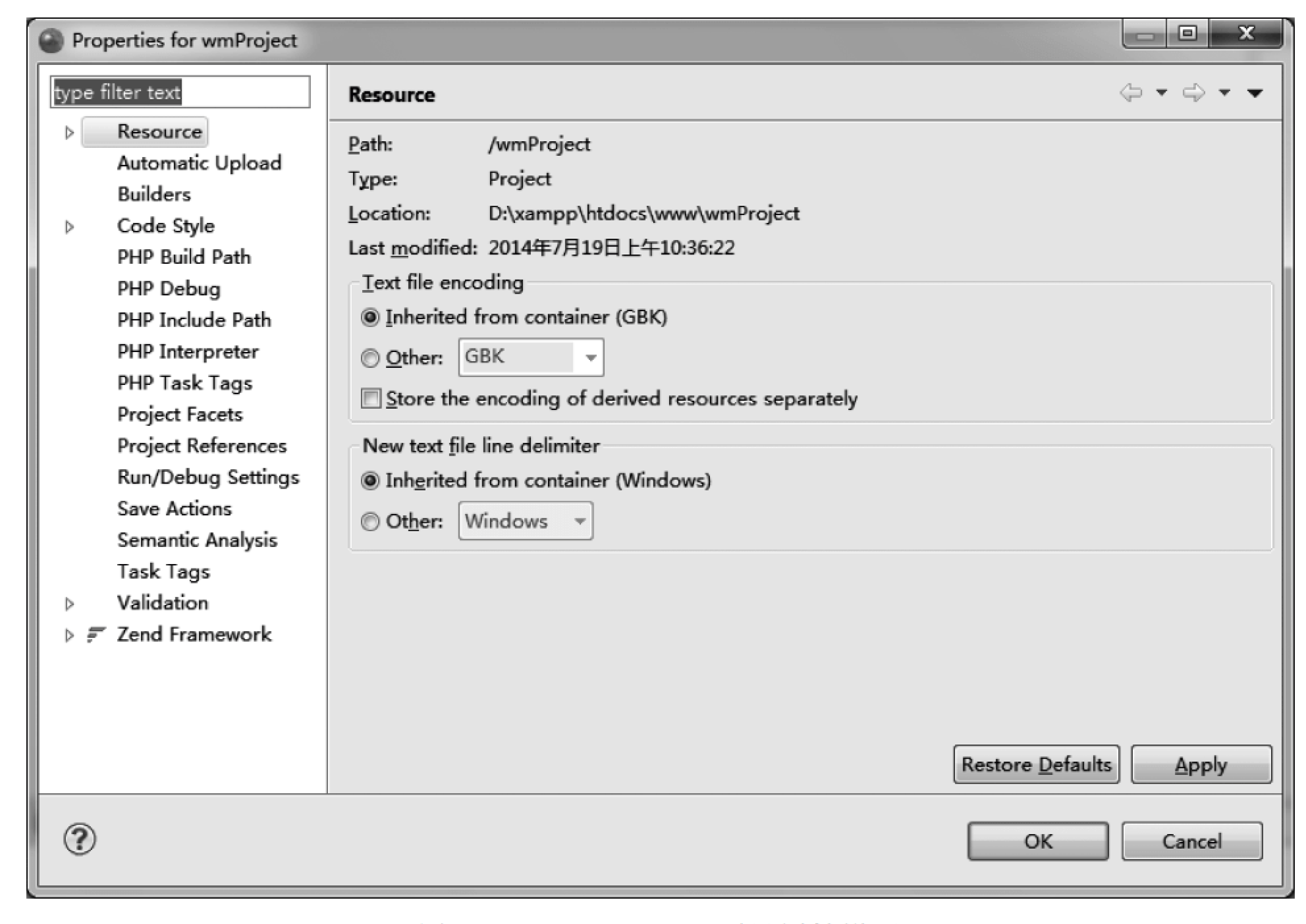


图 1.23　Zend Studio 项目属性设置

5. 查看源代码

在 Zend Studio 集成开发环境的代码编辑区用鼠标指向某一个类，就可以在提示框中查看该类的一些基本信息。如果要查看源代码，单击提示框的跳转图标即可，如图 1.24 所示。在该图中用鼠标指向类 Zend_Form，就会出现 Zend_Form 类的基本信息，单击下面的显示图标，就可以在该编辑区中打开 Zend_Form 类的源文件，这对项目开发是非常有好处的。

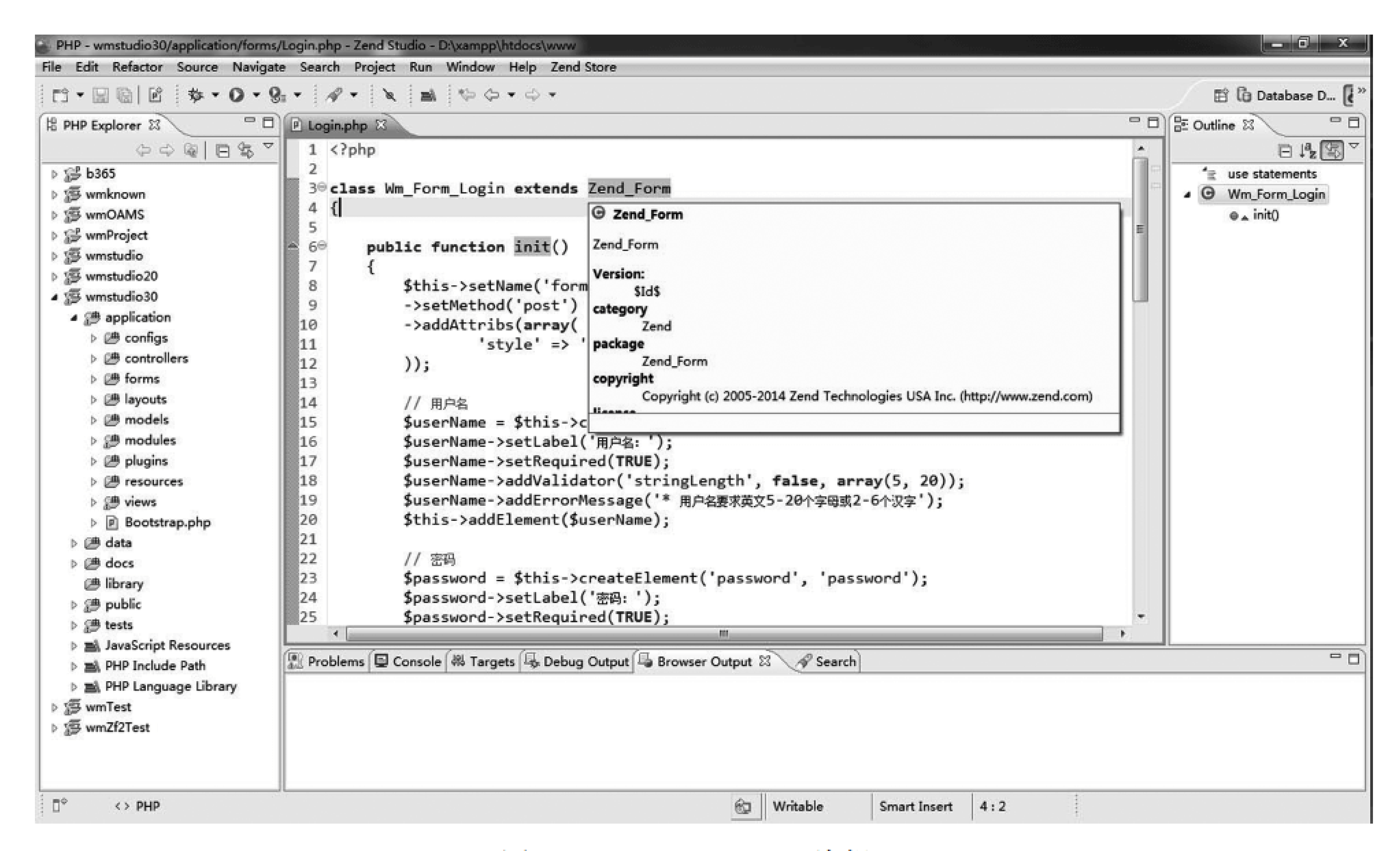


图 1.24　Zend Studio 编辑区

6. **查看类的变量及方法**

面向对象的编程主要是类的设计与方法的使用。对于初学者而言，最大的困难在于不知道类中到底有些什么样的方法，也不清楚方法的访问权限及调用后返回的数据类型。在 Zend Studio 集成开发环境中可以很轻松地解决这些问题。

如上所述，当打开 Zend_Form 类以后，在 Zend Studio 的 Outline 窗口中就会显示该类的属性与方法，如图 1.25 所示。单击 Outline 窗口中的某个方法，就可以在编辑区中打开该方法的源文件。

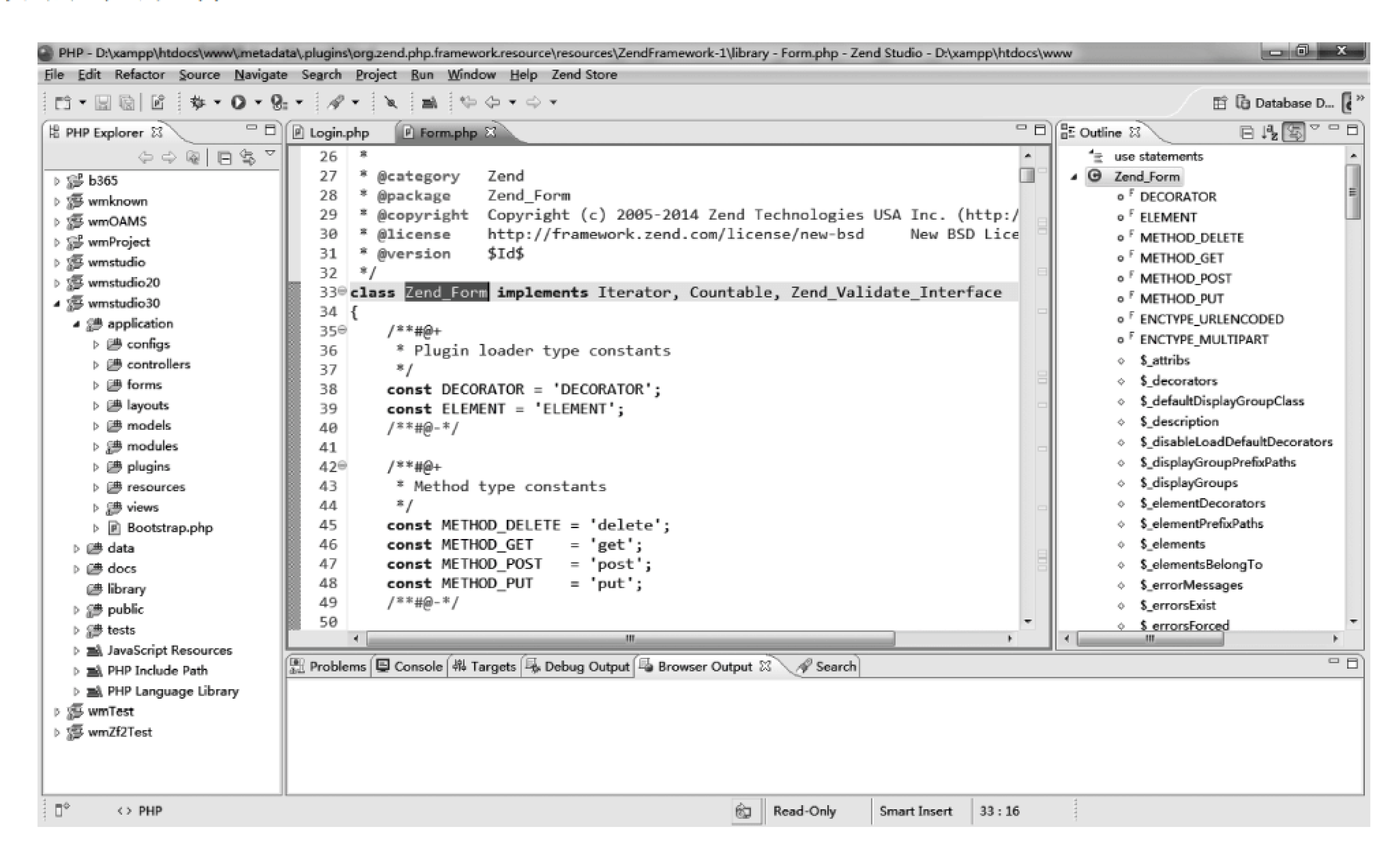


图 1.25 Zend Studio 中类信息的显示

7. **文件的复制与删除**

Zend Studio 中对文件的更名、复制与删除操作可以在项目工作区中直接进行。例如，要把一个图片文件复制到项目中的 images 文件夹，就可以先将该文件复制到剪切板上，然后直接单击工作区项目中的 images 文件夹，右键粘贴。如果要删除项目中的某个文件或文件夹，只要选中后右键删除，或选中后用键盘上的 Delete 键删除即可。需要特别提醒的是，这里的删除操作是直接从硬盘上删除，不会放入回收站中，也没有倒退键撤销该操作，这一点请大家务必注意。

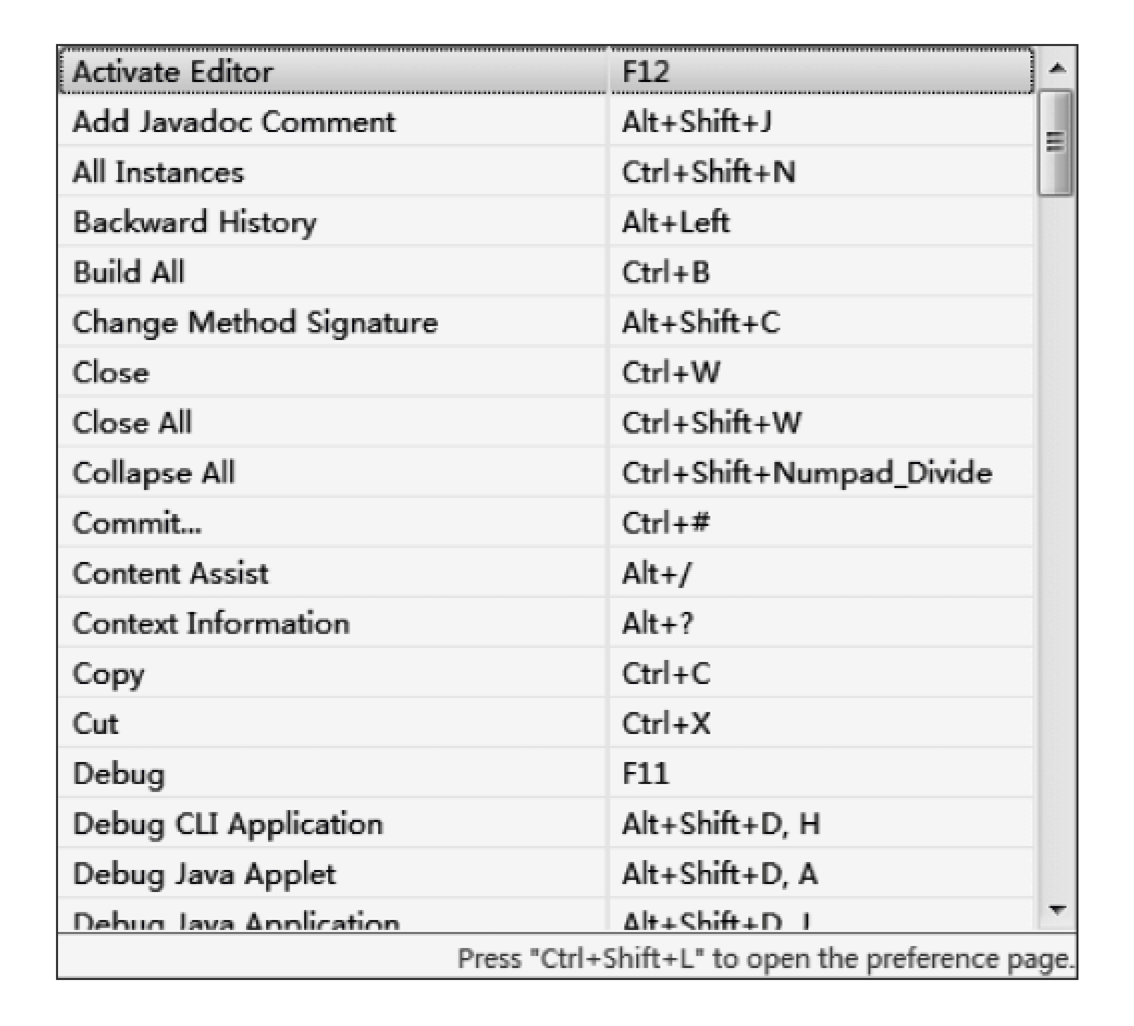
8. **常用快捷键**

采用快捷方式进行操作能大大地提高开发效率。菜单项的快捷方式可以在菜单栏中查看到，也可以通过组合键 Ctrl+Shift+L 查询，如图 1.26 所示。注意，不同版本的 Zend Studio 的快捷方式可能会有些差异。

1.3.2 Nodepad++代码编辑器

Notepad++是一款 Windows 环境下免费开源的代码编辑器，支持 C、C++、Java、C#、XML、HTML、PHP、JavaScript 等多种语言。

Notepad++是一套非常有特色的自由软件的纯文字编辑器，有完整的中文化接口及支



Activate Editor	F12
Add Javadoc Comment	Alt+Shift+J
All Instances	Ctrl+Shift+N
Backward History	Alt+Left
Build All	Ctrl+B
Change Method Signature	Alt+Shift+C
Close	Ctrl+W
Close All	Ctrl+Shift+W
Collapse All	Ctrl+Shift+Numpad_Divide
Commit...	Ctrl+#
Content Assist	Alt+/
Context Information	Alt+?
Copy	Ctrl+C
Cut	Ctrl+X
Debug	F11
Debug CLI Application	Alt+Shift+D, H
Debug Java Applet	Alt+Shift+D, A

Press "Ctrl+Shift+L" to open the preference page.

图 1.26　Zend Studio 常用的快捷键

持多国语言撰写的功能(UTF8 技术)。它的功能比 Windows 中的 Notepad 强大,除了可以用来制作一般的纯文字说明文件外,它还十分适合作为的撰写计算机程序的编辑器。Notepad++不仅有语法高亮度显示功能,也有语法折叠功能,并且支持宏以及扩充基本功能的外挂模块。

Notepad++编辑器界面如图 1.27 所示,其主要功能如下:

- 内置多达 27 种语法高亮度显示,包括各种常见的源代码、脚本,能够很好地支持 .nfo 文件的查看、支持自定义语言。
- 可自动检测文件类型,根据关键字显示节点,节点可自由折叠/打开,还可显示缩进引导线,代码显示层次感强。

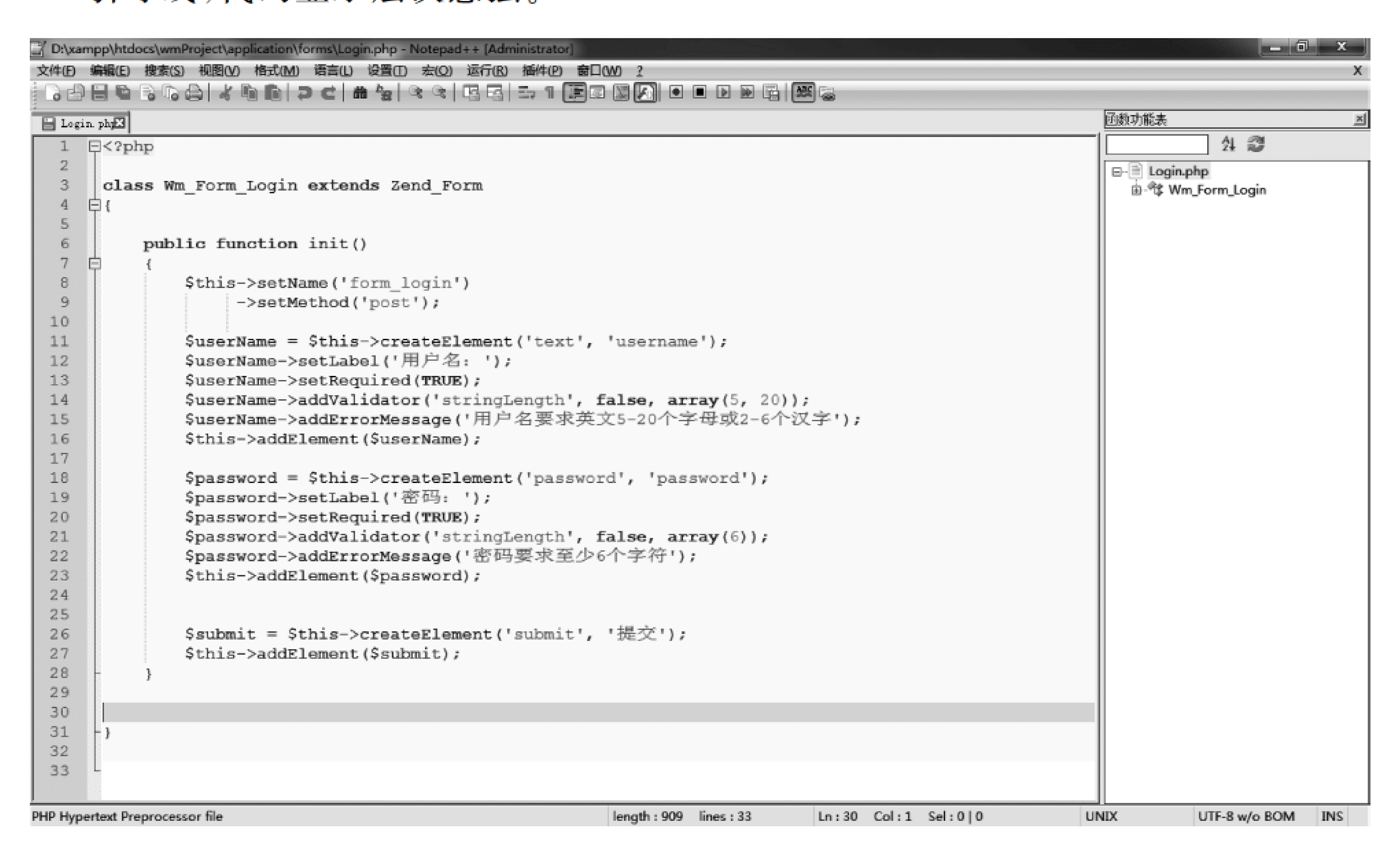


图 1.27　Notepad++编辑器

- 可打开双窗口，在分窗口中又可打开多个子窗口，可以调整显示比例。
- 提供了一些有用工具，例如邻行互换位置、宏功能等。
- 可显示选中文本的字节数，而不是一般编辑器所显示的字数。
- 正则匹配字符串及批量替换。
- 强大的插件机制，扩展了编辑功能。

下面简单介绍 Notepad++编辑器的常用操作，这里以 6.6.7 版本为例。

1. 编辑器语言设置

Notepad++编辑器安装完毕后，选择菜单栏中的“设置”|“首选项”菜单项，弹出“首选项”对话框。切换到“常用”选项卡，在“界面语言”下拉列表中选择“中文简体”，将编辑器界面语言从英文转变为简体中文。

2. 语言格式设置

Notepad++默认提供了许多语言的代码高亮功能，如果程序没有自动识别，选择菜单栏中的“设置”|“语言格式设置”菜单项，弹出“语言格式设置”对话框，选择语言类型后，即可设置语言显示的背景颜色及字体样式等。

3. 代码提示

在 Notepad++中，默认的代码自动完成快捷键是 Ctrl＋Enter。例如在 PHP 文件中输入 c，然后按 Ctrl＋Enter，就会显示出来代码提示。通过快捷键有时不太方便，可以在“首选项”对话框的“自动完成”选项卡中选择“所有输入均启用自动完成”复选框、“输入时提示函数”复选框，来设置自动提示功能。

4. 书签功能

书签的设置在如图 1.27 所示的中间编辑区中进行。在编辑区中的行号旁边单击，或者是在定位的行上按 Ctrl＋F2 快捷键，会发现书签栏中多出了一个蓝色圆点，这个蓝色圆点即为书签标记。再次单击书签标记，或者是在书签标记行按 Ctrl＋F2 快捷键，可以取消书签。

当光标移到编辑区中的其他位置时，按下 F2 功能键，即可回到标签行。在设置有多个书签的文件中，通过功能键 F2 移动光标到上一个书签，通过 Shift＋F2 组合键将光标移动到下一个书签。

5. 列编辑功能

Notepad++的列编辑就是在光标所在列插入文本或者数字，可以通过选择“编辑”|“列块模式”或“列块编辑”菜单项完成，也可以通过按快捷键 Alt＋C 来完成。

例如，要在图 1.27 所示的代码前加上自己的行号，可以按下列步骤进行操作：

(1) 将光标定位到代码的第 1 行第 1 列。

(2) 按快捷键 Alt＋C，弹出“列编辑”对话框。

(3) 在“列编辑”对话框中选择“插入数字”，设置“初始值”与“增量”。

(4) 单击该对话框中的“确定”按钮，完成行号的设置，如图 1.28 所示。

注意，这里设置的行号布满文件中光标所在行下面的所有代码行。如果要对选择的行进行列操作，先选择“编辑”|“列块模式”菜单项、启动列编辑模式，然后用“Alt＋鼠标左键”或 Alt＋Shift＋Arrow Key 选择要操作的行，最后在“列编辑”对话框中进行设置。

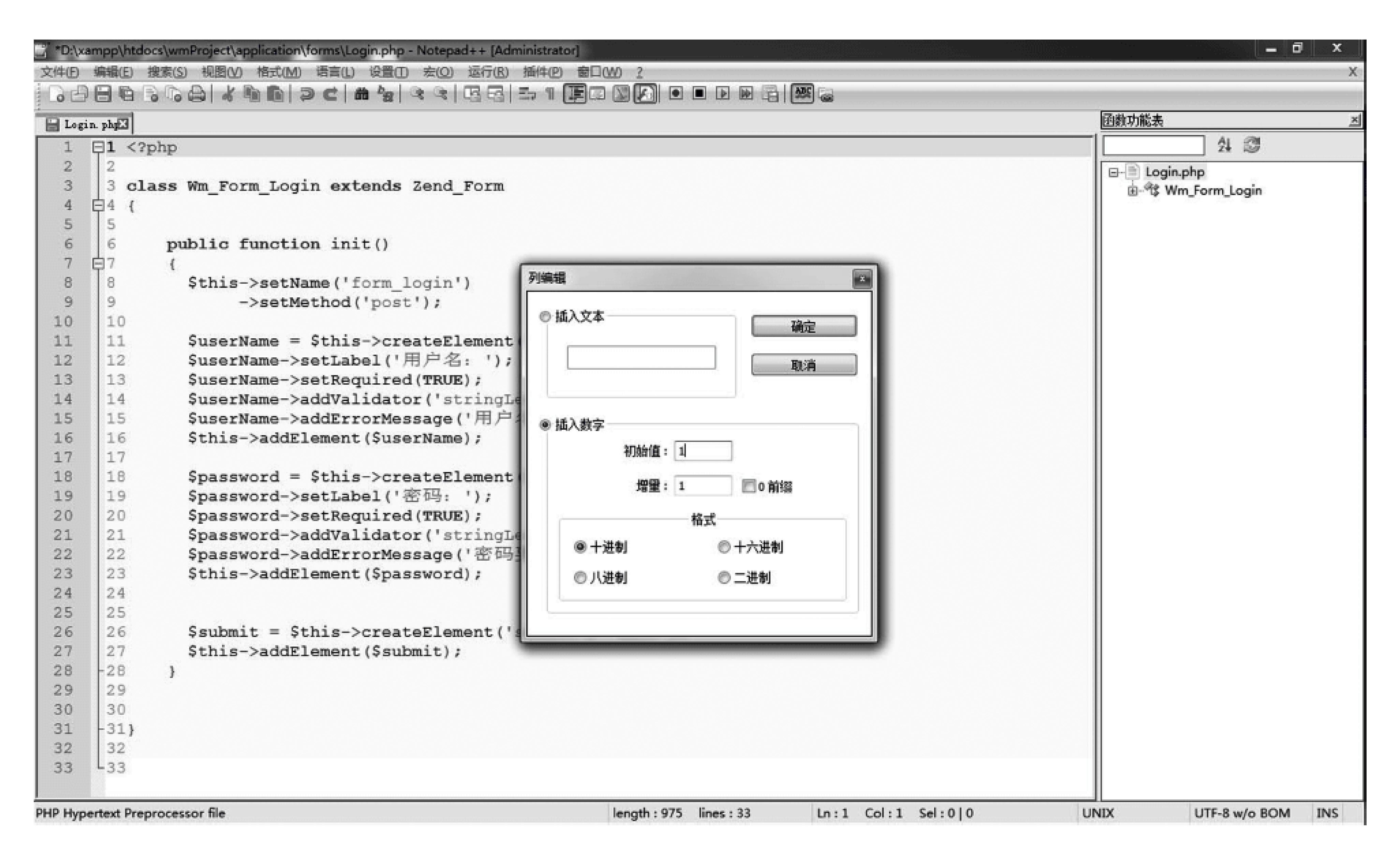


图 1.28　Notepad++列编辑

6. 颜色标记

颜色标记就是给选定的文本设置定义的格式。其使用方法非常简单，选中需要标记的文本，右键选择标记的格式即可，用同样的方法也可以清除设置的格式。

7. Nppexport 插件

Nppexport 是 Notepad++默认安装的插件，通过这款插件可以方便地导出着色以后的代码。其功能通过“插件”|Nppexport 下的子菜单项完成。

8. Converter 插件

Converter 也是一款 Notepad++默认安装的插件，通过这款插件可以方便地进行各种数制的转换。其功能通过“插件”|Converter 下的子菜单项完成。

1.3.3 技术文档

在 Zend Framework 项目开发过程中，用户会经常查阅一些技术文档，以了解某个函数的调用方法或某个类的结构等，这就需要在开发前做好技术资料的收集与整理工作。

由于 Zend Framework 项目涉及的知识面非常广，因此，在开发过程中需要查阅的资料会非常多，另外，开发者的知识结构、专业水平不同，查阅的技术文档类型也会有所不同，作者认为下面两个手册应该是大家必须要准备的。

1. PHP 使用手册

通过 PHP 使用手册可以查询到 PHP 函数的定义及调用方法，也可以了解随着 PHP 版本的升级而被废弃的函数。当然，该手册中的某些示例代码也可以直接复制到项目中。其界面如图 1.29 所示。

2. Zend Framework 使用手册

Zend Framework 使用手册是学习 Zend Framework 必不可少的技术资料，是开发人员进行项目开发的得力助手，如图 1.30 所示。通过该手册不仅可以查询到 Zend Framework

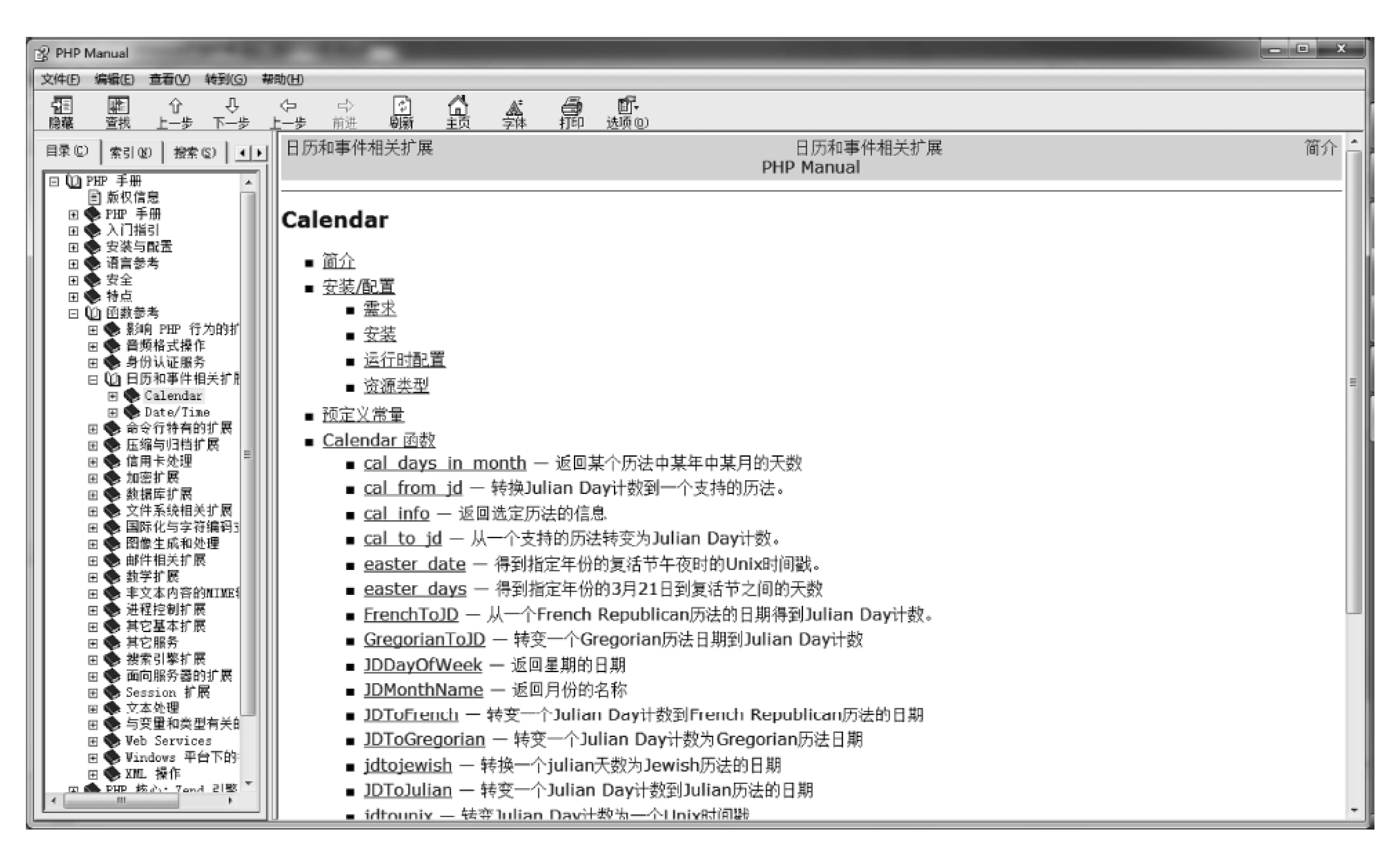


图 1.29　PHP 使用手册

的组件结构，了解其功能；还可以参考其给出的示例程序，以及用户在使用过程中碰到问题后所给出的解决方案。大家需要注意的是，手册对应着不同的 PHP 或 Zend Framework 版本，各版本之间会有一些差别。

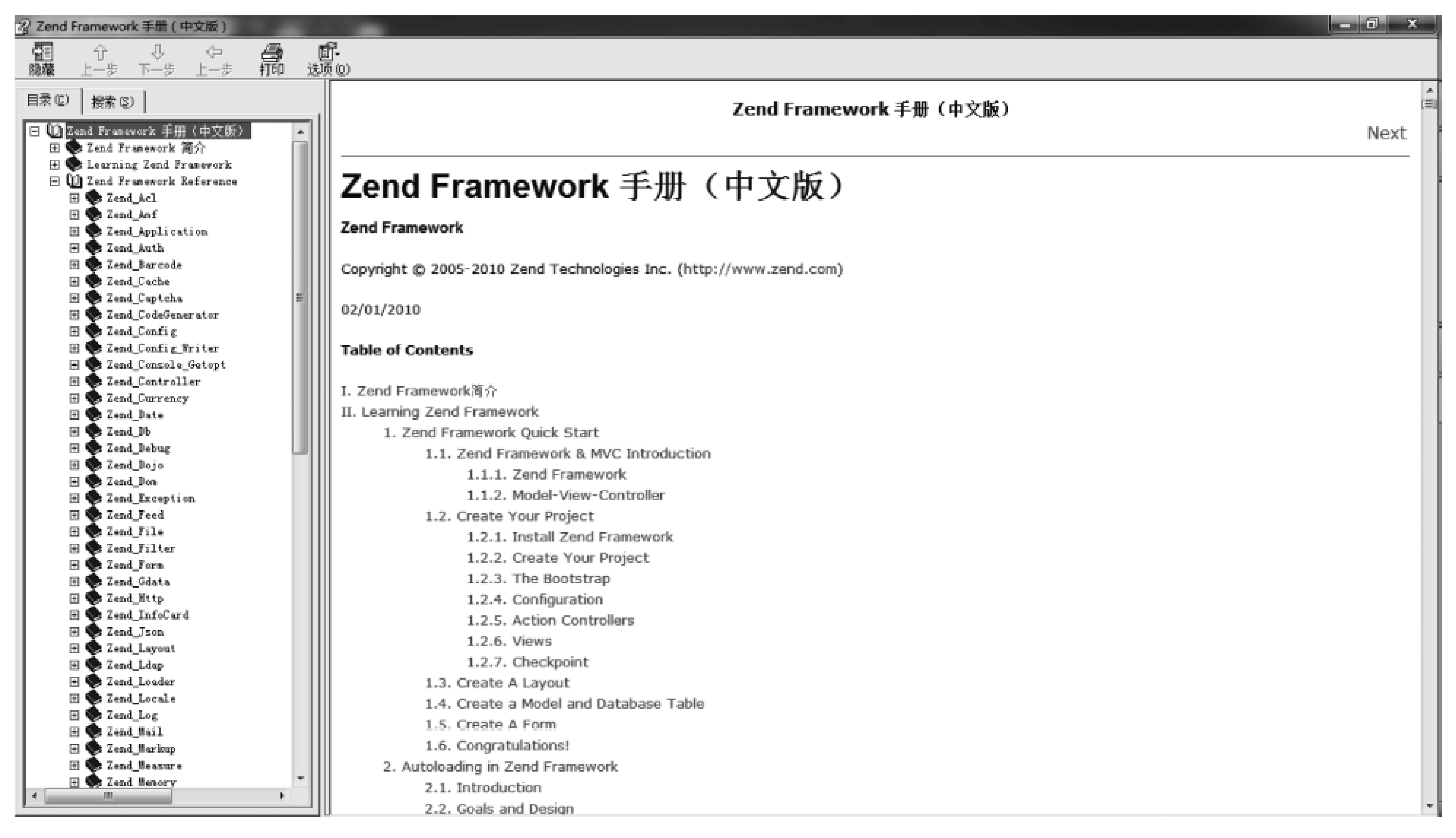


图 1.30　Zend Framework 使用手册

1.4　本章小结

本章在对 PHP 的 Zend Framework 框架技术概述的基础上，重点介绍其应用开发的环境搭建与配置过程以及 Zend Studio 与 Notepad++两款开发工具的使用方法。除本章介绍



的开发环境与工具之外，还有许多其他的配置方案，大家可以根据自身情况自由选择。Zend Framework 应用是跨平台的，可以在绝大多数支持 PHP 环境的平台上运行。

本章内容是为初学者提供的，已经有了 PHP 项目开发经验的读者，在了解了 Zend Framework 的常用组件、安装过程之后，可以略过本章的其他内容，直接进入下一章的学习。

第 2 章　Zend Framework 结构及原理

Zend Framework 框架是基于 MVC 模式的，其项目有相对固定的目录结构。这种目录结构形式是由 Zend Framework 框架的路由原理与文件搜索机制决定的，所以在进行项目开发时要严格遵循这种格式，相对固定的结构有利于项目后期的功能扩展与维护。

本章介绍 Zend Framework 项目的创建及运行原理。

2.1　Zend Framework 项目的创建和结构

第 1 章创建了 Zend Framework 的开发环境，并成功安装、运行了一个使用 Zend Framework 组件 Zend_Date 的 PHP 文件 Date.php，在浏览器页面中正确地输出了当前日期。下面开始创建我们的第一个基于 Zend Framework 框架的项目。

2.1.1　Zend Framework 项目的创建

创建 Zend Framework 项目有多种方法，可以根据 Zend Framework 框架的特点手工构建，也可以用 Zend Framework 自带的命令工具创建，还可以用 PHP 的集成开发环境 Zend Studio 自动创建。

1. 使用集成开发环境

在新的 Zend Studio 集成开发环境的版本中嵌入了 Zend Framework 库文件及开发工具，因此，可以通过应用程序向导自动生成 Zend Framework 应用程序。

(1) 打开 Zend Studio 集成开发环境，选择工作区为 D:\xampp\htdocs 目录，然后选择 File|New|Local PHP Project 菜单项，弹出如图 2.1 所示的 New Local PHP Project 对话框。

(2) 在 Project Name 文本框中输入项目名称，这里案例项目名为 wmProject；在 Content 单选组中选择 Zend Framework；在 Version 下拉列表框中选择 Zend Framework 框架版本，本书项目选择 Zend Framework 1.12.5，如图 2.2 所示。当然，也可以通过 Version 下拉列表框中的 Other 选项打开如图 2.3 所示的对话框加载 Zend Framework 的最新版本或自己下载的其他版本的库文件。

(3) 单击图 2.2 所示的属性页对话框中的 Finish 按钮，即可生成如图 2.4 所示的 Zend Framework 项目。

2. 使用 Zend Framework 命令工具

Zend Framework 自带了 ZF Tool 命令工具，用来快速创建项目、控制器、模型、视图以及表单等资源。在下载 Zend Framework 后的压缩包里有一个 bin 文件夹，其中有 zf.bat、zf.php 和 zf.sh 3 个文件，zf.bat 就是 Windows 系统下使用的命令文件。

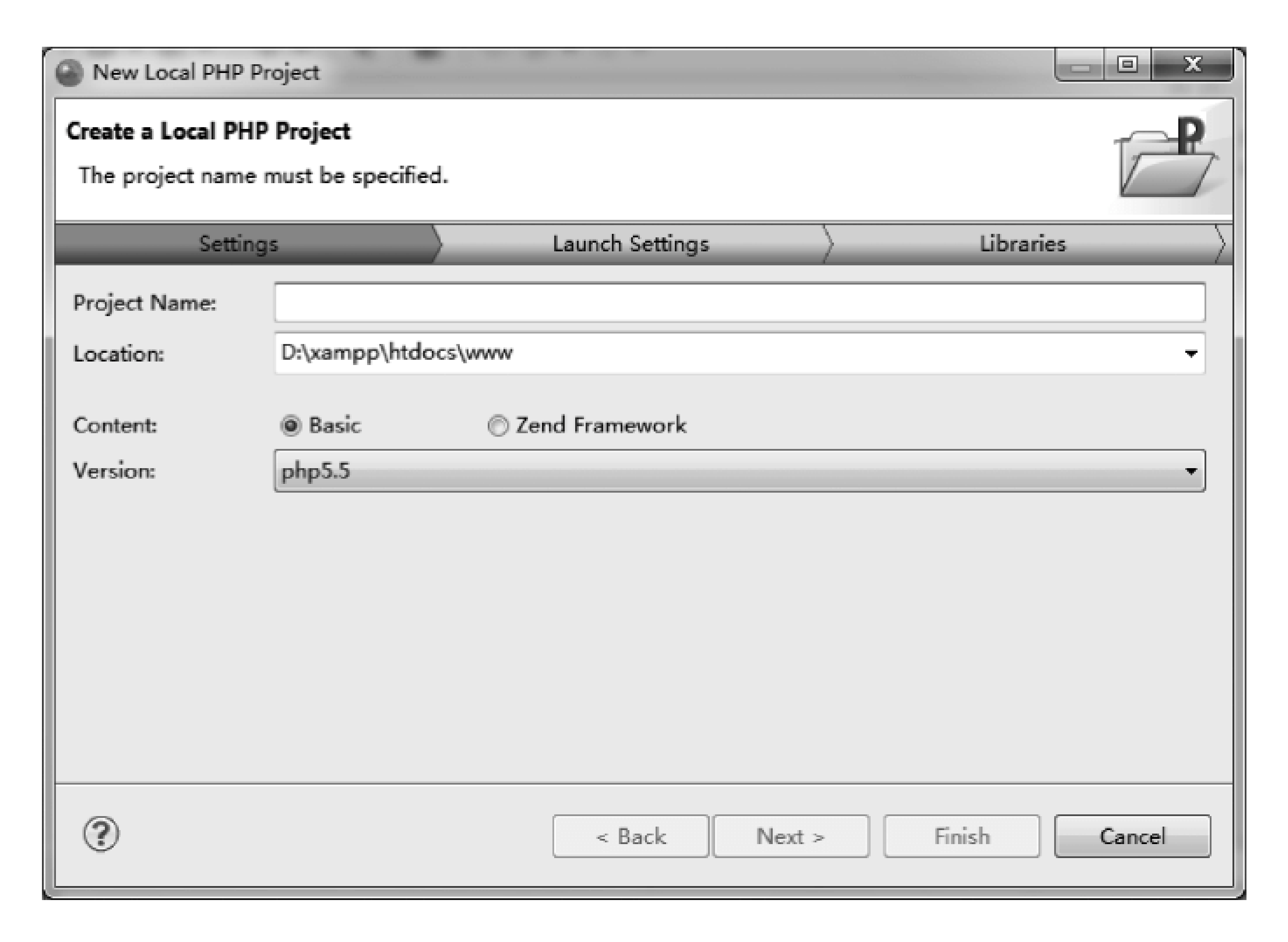


图 2.1　新建本地 PHP 项目对话框

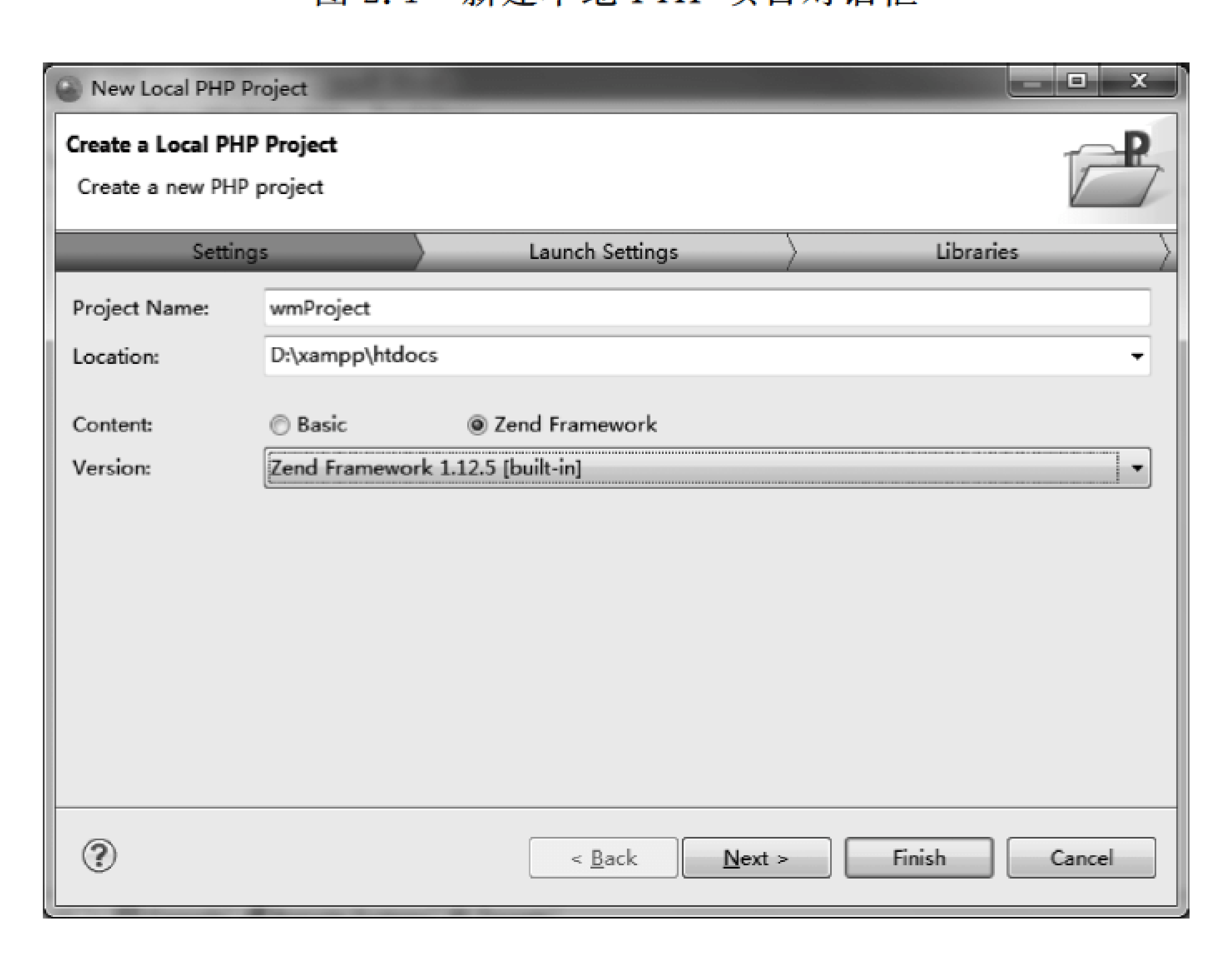


图 2.2　新建 PHP 项目信息

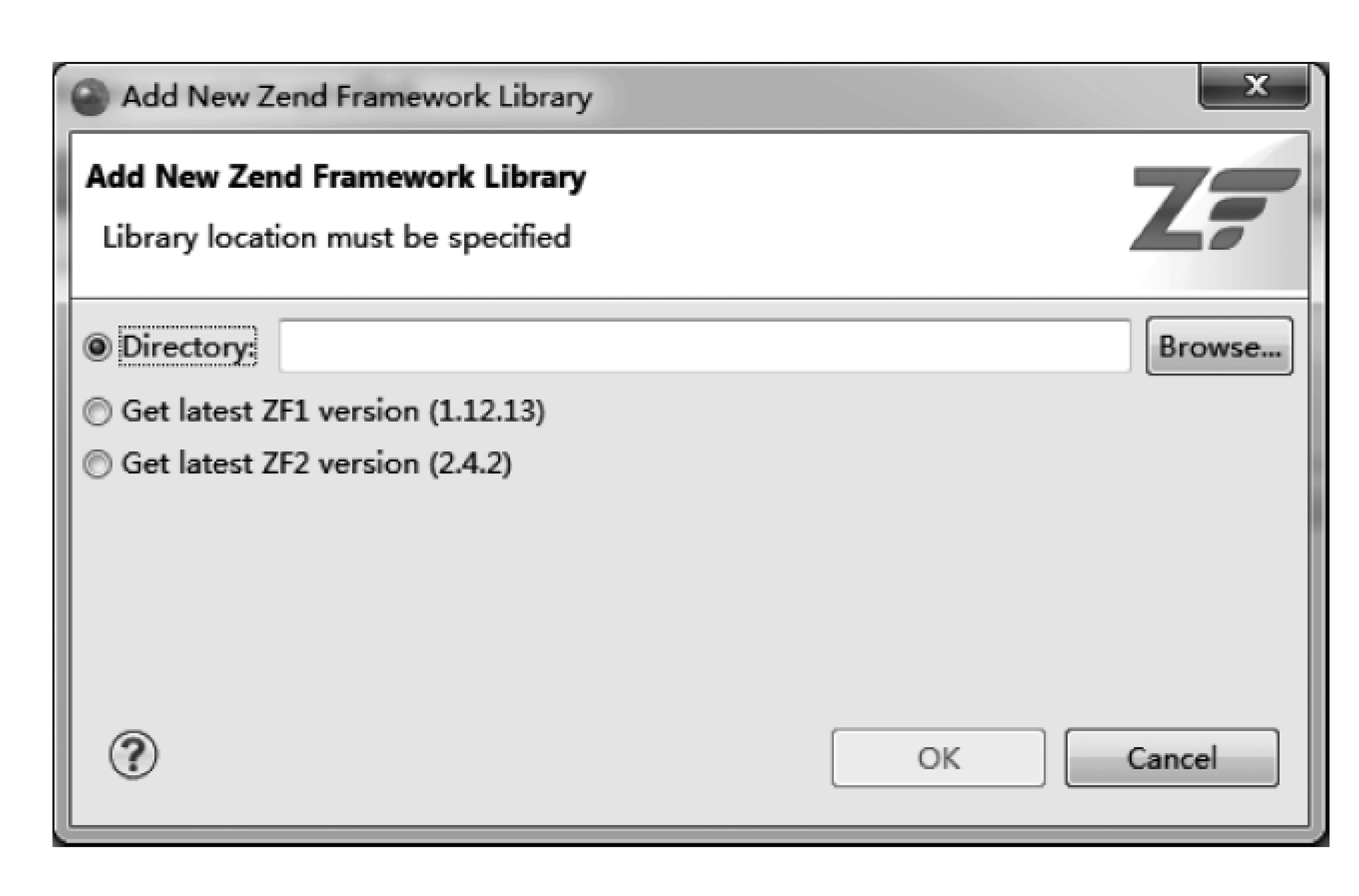


图 2.3　添加 Zend Framework 其他版本

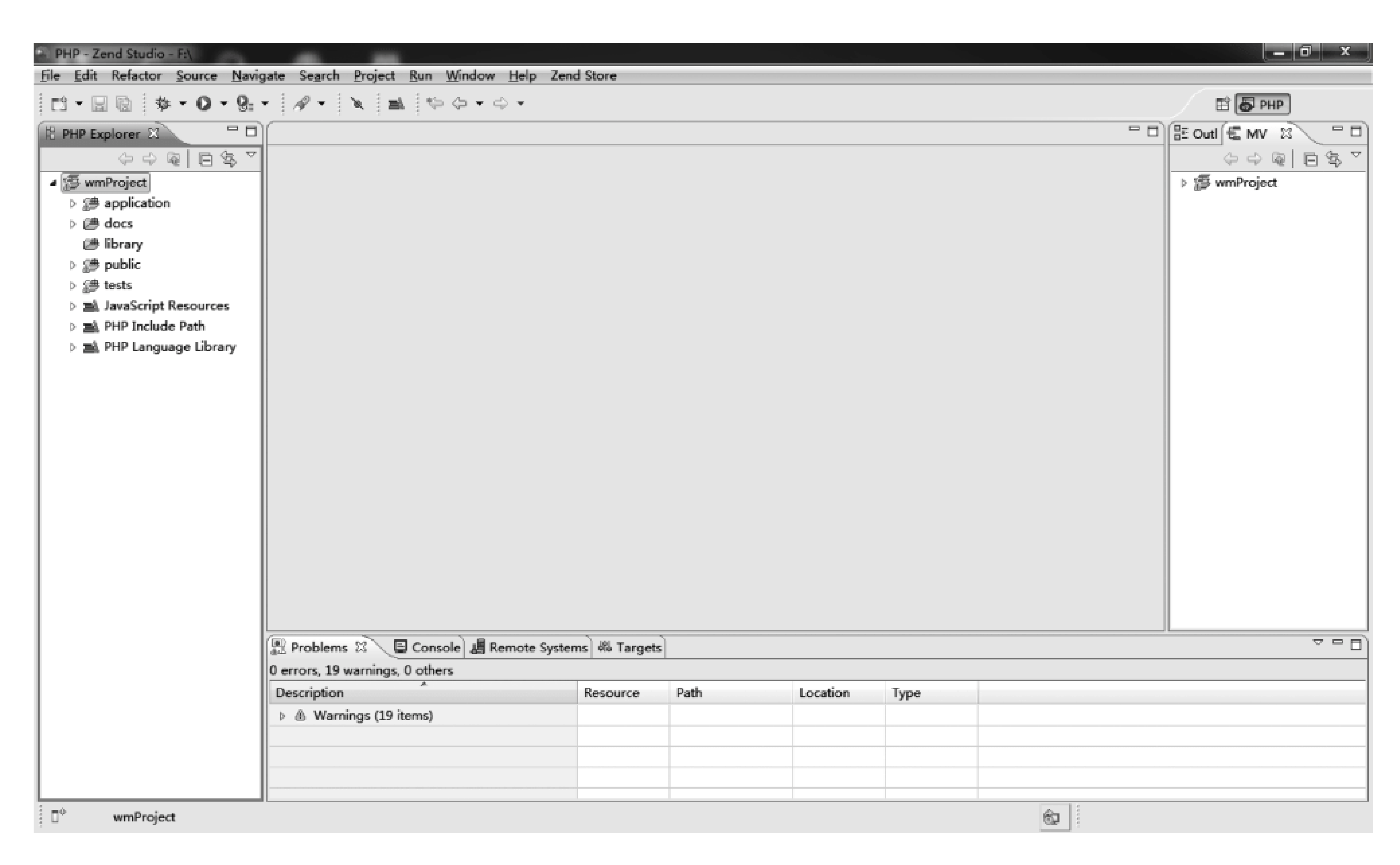


图 2.4　新建 Zend Framework 项目界面

1）直接使用 ZF Tool 工具

打开 Windows DOS 命令窗口，将目录切换到下载的 Zend Framework 的 bin 文件夹下，输入命令 zf. bat create project f:/wmProject，即可在 F 盘根目录下创建项目 wmProject，如图 2.5 和图 2.6 所示。

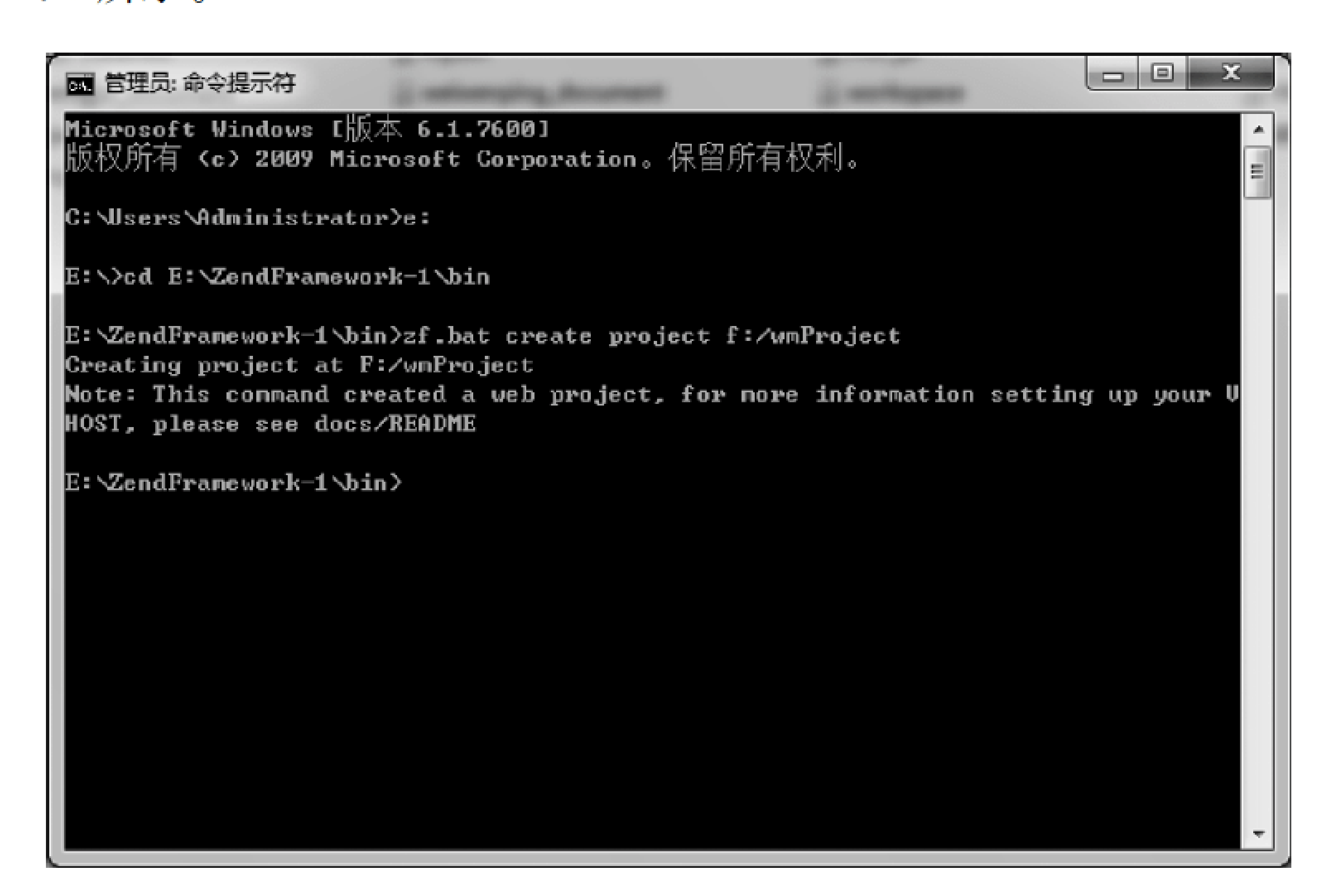


图 2.5　新建 Zend Framework 项目命令窗口

2）先安装 ZF Tool 工具再使用

使用上述命令方式创建 Zend Framework 项目是直接在下载的 Zend Framework 文件夹中进行操作的，不是很方便，所以一般会将命令文件添加到系统的搜索路径当中，这样在用命令创建项目时就可以直接在拟建项目所在的目录中进行操作了。

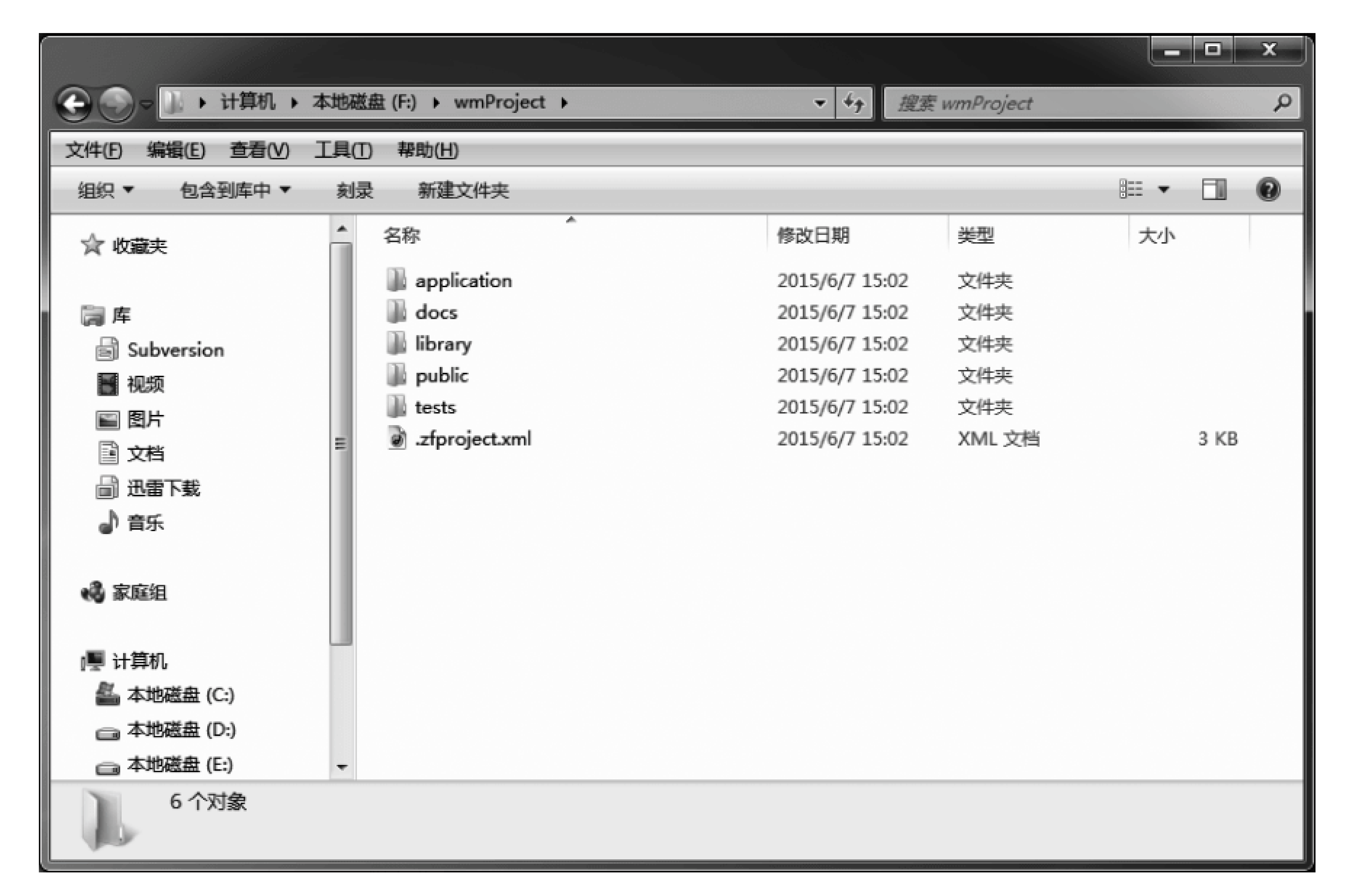


图 2.6　用命令方式新建 Zend Framework 项目目录

安装 ZF Tool 工具的步骤如下：

(1) 在 C:\Program Files 目录下创建 Zend 新文件夹。

(2) 将下载的 Zend Framework 解压包中的 bin 文件夹复制到 Zend 文件夹中。

(3) 将 C:\Program Files\Zend\bin 目录添加到系统环境变量 path 中。

在安装好 ZF Tool 命令工具后，打开如图 2.5 所示的 Windows 命令窗口，将目录切换到 F 盘根目录下，输入命令：

```
zf create  project  wmProject
```

即可在 F 盘根目录下创建一个名为 wmProject 的 Zend Framework 项目。

3. 使用 Zend Studio 集成环境中的 ZF Tool 工具

打开 Zend Studio 集成开发环境，选择 Project|Zend Tool 菜单项，弹出如图 2.7 所示的命令窗口，输入命令：

```
zf create  project  wmProject
```

即可在 Zend Studio 集成开发环境的项目工作区中创建一个名为 wmProject 的 Zend Framework 项目。这时，如果项目工作区的目录对应 D:\xampp\htdocs 目录，项目 wmProject 就被创建在了本地服务器上。

2.1.2　Zend Framework 命令

Zend Framework 提供的命令工具为我们的应用开发提供了方便，安装 Zend Framework 的 ZF Tool 工具后，在 Windows 的命令窗口中直接输入：

```
zf ?
```

命令，就会显示 ZF Tool 命令的所有说明和语法提示，如图 2.8 所示。

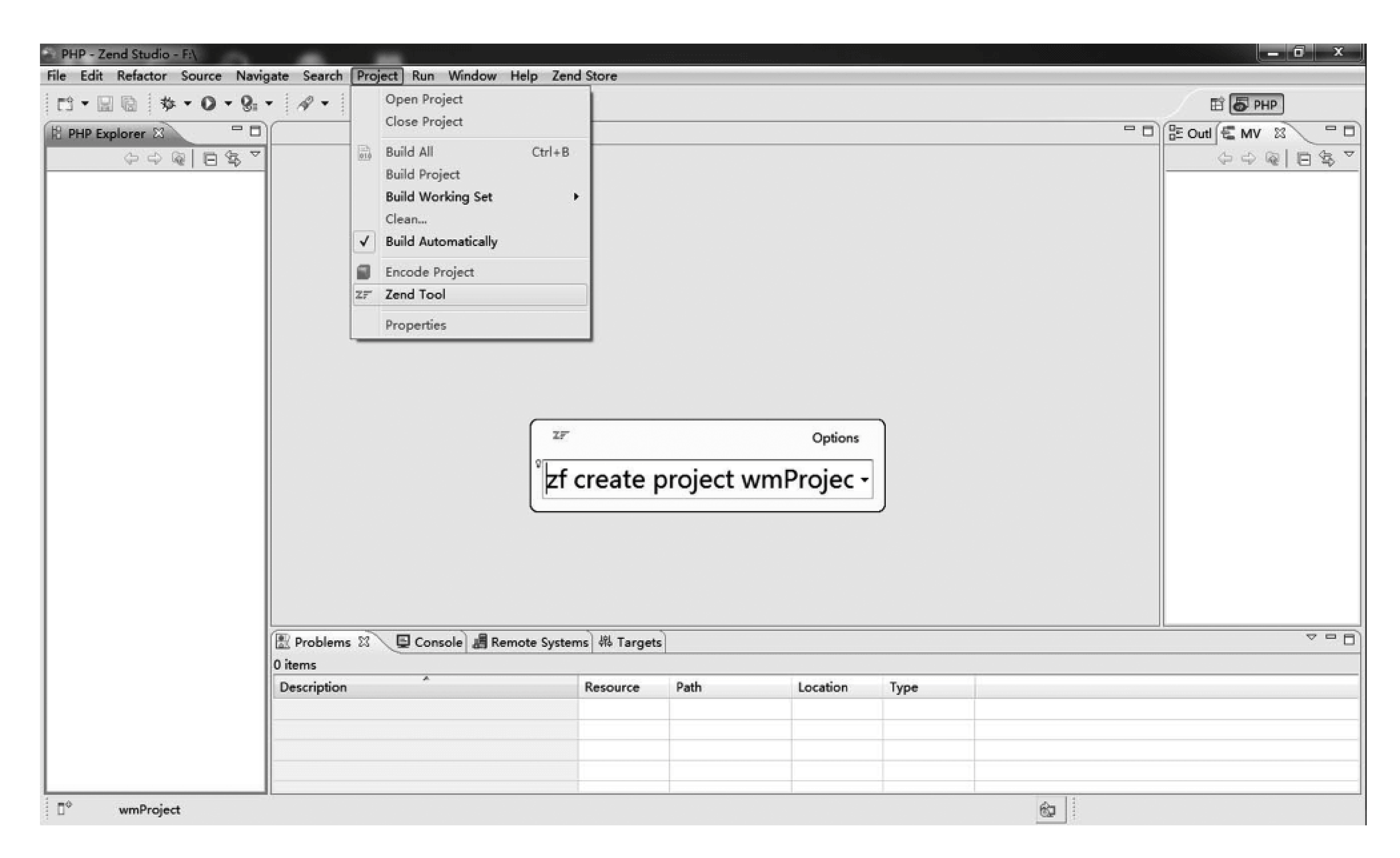


图 2.7　输入命令

关于 ZF Tool 命令，这里不给出详细的语法说明，大家先通过图 2.8 所示的方法了解一下其概貌。在接下来的项目开发过程中会频繁地使用这些命令，到时再详细讲解。

2.1.3　Zend Framework 项目结构

打开上面创建的 Zend Framework 项目 wmProject，其项目结构如图 2.9 所示。

管理员: 命令提示符

```
Microsoft Windows [版本 6.1.7600]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>zf ?
Zend Framework Command Line Console Tool v1.12.5
Usage:
    zf [--global-opts] action-name [--action-opts] provider-name [--provider-opt
s] [provider parameters ...]
    Note: You may use "?" in any place of the above usage string to ask for more
 specific help information.
    Example: "zf ? version" will list all available actions for the version prov
ider.

Providers and their actions:
  Version
    zf show version mode[=mini] name-included[=1]
    Note: There are specialties, use zf show version.? to get specific help on t
hem.

  Config
    zf create config
    zf show config
    zf enable config
    Note: There are specialties, use zf enable config.? to get specific help on
them.
```

图 2.8　显示 ZF Tool 命令的所有说明和语法提示

PHP Explorer

```
wmProject
  application
    configs
    controllers
    models
    views
      helpers
      scripts
        error
        index
          index.phtml
    Bootstrap.php
  docs
    README.txt
  library
  public
    .htaccess
    index.php
  tests
    application
    library
    bootstrap.php
    phpunit.xml
```

图 2.9　项目结构

这是 Zend Framework 的基本目录结构，当然还可以根据具体情况添加一些新的目录与文件。例如，在 public 目录下添加存放图片文件的 images 目录、存放样式文件的 CSS 目录以及存放 JavaScript 脚本的 JS 目录等。Zend Framework 目录中各文件夹的作用见表 2.1。

表 2.1 Zend Framework 项目的文件夹及说明

文　件　夹	说　　明
application	应用程序主目录，包含应用程序全部文件，一般为非公开文件，例如配置文件等
application\configs	项目配置文件目录
application\modules	多模块目录，例如 admin(后台管理)、default(默认前台管理)等。由于在上面的项目中还没有创建模块，在图 2.9 所示的目录结构中该目录暂时不存在(第 7 章介绍)
application\controllers	控制器目录
application\models	模型目录
application\views	视图目录
application\views\helpers	存放“动作助手”(action helper)的目录
Bootstrap. php	应用程序启动文件，用于引导应用程序、注册并初始化组件
docs	文档和说明文件
library	库文件目录。应用程序引用的第三方类库和自己编写的类库在这里自动加载
public	应用程序的公开文件目录，也是项目的根目录，存放可以被用户访问的文件，例如 JavaScript、CSS 和图片等，项目的主页面文件 index. php 也存放在这里
tests	存放测试文件。如果系统安装了 PHP Unit，则在使用 ZF Tool 创建控制器方法时会自动在该文件夹下创建测试文件

2.2 Zend Framework 项目的运行

上面用不同的方法在不同的目录中创建了 Zend Framework 项目 wmProject，并且了解了它的目录结构及首页文件 index. php 的存放位置，下面运行通过 Zend Studio 在 D:\xampp\htdocs 中创建的 wmProject 项目。

1. 直接运行

(1) 打开 XAMPP 的控制面板，启动 Apache 服务器。

(2) 启动 IE 或其他浏览器。

(3) 在浏览器地址栏中输入 http://localhost/wmProject/public，即可看到如图 2.10 所示的主页面效果。由于页面中的背景图片取自 Zend 公司网站，所以如果计算机没有连接外网，将看不到背景图片。

2. 通过虚拟主机运行

在第 1 章中已经创建了名为 wmoams. com 的虚拟主机，其根目录为 D:\xampp\htdocs\wmProject\public。这就是说，http://wmoams. com 即为我们创建的 Zend Framework 项目 wmProject 的系统域名。

启动 Apache 服务器，在浏览器中输入 http://wmoams. com，即可看到如图 2.10 所示的欢迎页面效果。另外，在浏览器中输入 http://wmoams. com/index 或 http://wmoams. com/index/index，也会看到同样的页面效果。

正常情况下，Zend Framework 应用的访问 URL 应该为“域名＋模块名＋控制器名＋

动作名”，那么为什么上述 URL 省略了部分内容也可以访问呢？这是因为在 Zend Framework 中默认模块名(Default)、默认控制器名(Index)和默认首页动作名(index)都是可以省略的。

Zend Framework 的运行应当采用方法 2 的方式，这也是 Zend Framework 项目的标准运行方式。采用方法 1 虽然可以运行 Zend Framework 项目，但项目文件中的 URL 需要进行特别处理，例如采用 baseURL()函数获取 URL 等，这对初学者来说具有一定的难度。



图 2.10 Zend Framework 项目主页面

3. 改变视图内容

从表 2.1 可知，Zend Framework 项目的主页文件 index.php 存放在 public 目录下，打开该文件看一下里面的代码：

```
<?php
// Define path to application directory
defined('APPLICATION_PATH')
    || define('APPLICATION_PATH',realpath(dirname(__FILE__) . '/../application'));
// Define application environment
defined('APPLICATION_ENV')
    || define('APPLICATION_ENV',(getenv('APPLICATION_ENV') ? getenv('APPLICATION_ENV') :
'development'));
$paths = array(realpath(APPLICATION_PATH . '/../library'));
if (function_exists('zend_deployment_library_path') && zend_deployment_library_path('Zend
Framework 1')) {
        $paths[] = zend_deployment_library_path('Zend Framework 1');
}
$paths[] = get_include_path();
```

```
set_include_path(implode(PATH_SEPARATOR, $ paths));
/** Zend_Application */
require_once 'Zend/Application.php';
// Create application, bootstrap, and run
$ application = new Zend_Application(
    APPLICATION_ENV,
    APPLICATION_PATH . '/configs/application.ini'
);
$ application->bootstrap()
            ->run();
```

显然，这些代码并不是我们熟悉的 HTML 页面。但从图 2.10 所示的运行效果可以看到，浏览器显示的页面上既有图片又有文本，还有超级链接，内容还是非常丰富的。那么，这些页面元素到底是从哪里来的呢？

打开 application\views\scripts\index\index.phtml 文件，看一下里面的代码：

```
<style>
    a:link,
    a:visited
    {
        color: #0398CA;
    }
    span#zf-name
    {
        color: #91BE3F;
    }
    div#welcome
    {
        color: #FFFFFF;
        background-image: url(http://framework.zend.com/images/bkg_header.jpg);
        width:  600px;
        height: 400px;
        border: 2px solid #444444;
        overflow: hidden;
        text-align: center;
    }
    div#more-information
    {
        background-image: url(http://framework.zend.com/images/bkg_body-bottom.gif);
        height: 100%;
    }
</style>
<div id="welcome">
    <h1>Welcome to the <span id="zf-name">Zend Framework!</span></h1>
    <h3>This is your project's main page</h3>
    <div id="more-information">
        <p><img src="http://framework.zend.com/images/PoweredBy_ZF_4LightBG.png" /></p>
        <p>
            Helpful Links: <br />
            <a href="http://framework.zend.com/">Zend Framework Website</a> |
```

```
            <a href="http://framework.zend.com/manual/en/">Zend Framework Manual</a>
        </p>
    </div>
</div>
```

分析上面的代码，可知这里才是真正的主页页面。也就是说，访问项目的根目录中的public\index.php文件，结果呈现出来的是application\views\scripts\index\index.phtml文件中的内容。

将index.phtml文件中的代码全部删除，在页面中重新书写一行文本：

```
Welcome to http://www.wmstudio.net.cn
```

刷新页面，会看到首页变成了如图2.11所示的效果。

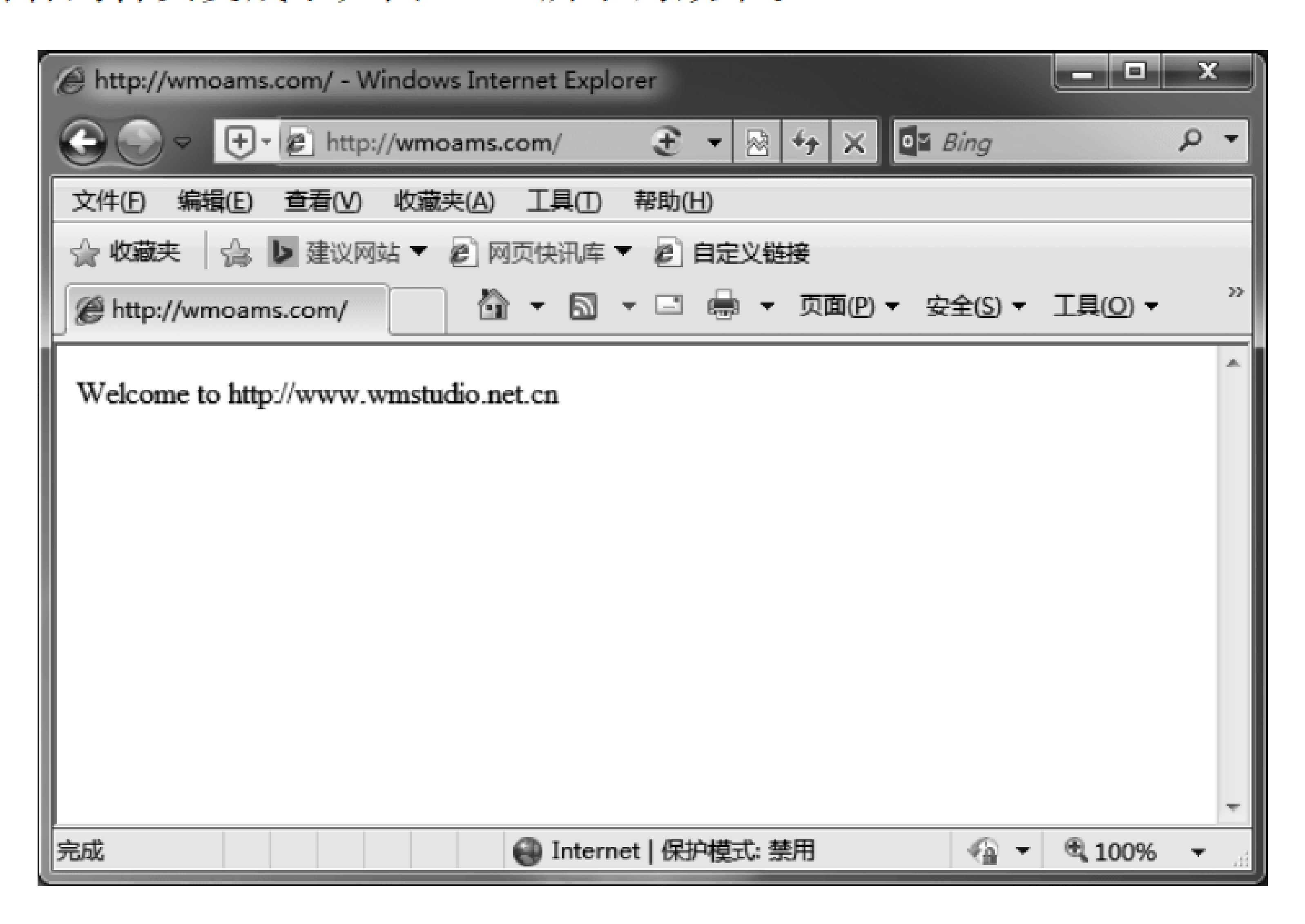


图2.11　修改页面内容后显示的效果

这说明，在浏览器中访问项目的根目录确实访问到了application\views\scripts\index目录下的index.phtml文件。另外，如果打开application\controllers\indexController.php文件，并在其方法indexAction中添加如下阴影部分代码：

```
<?php
class IndexController extends Zend_Controller_Action
{
    public function init()
    {
        /* Initialize action controller here */
    }
    public function indexAction()
    {
        //添加代码
        echo '<br />';
        echo 'welcome to Zend Framework IndexController';
    }
    public function mainAction()
```

```
        {
            // action body
        }
    }
```


刷新页面，会看到首页变成了如图 2.12 所示的样子。

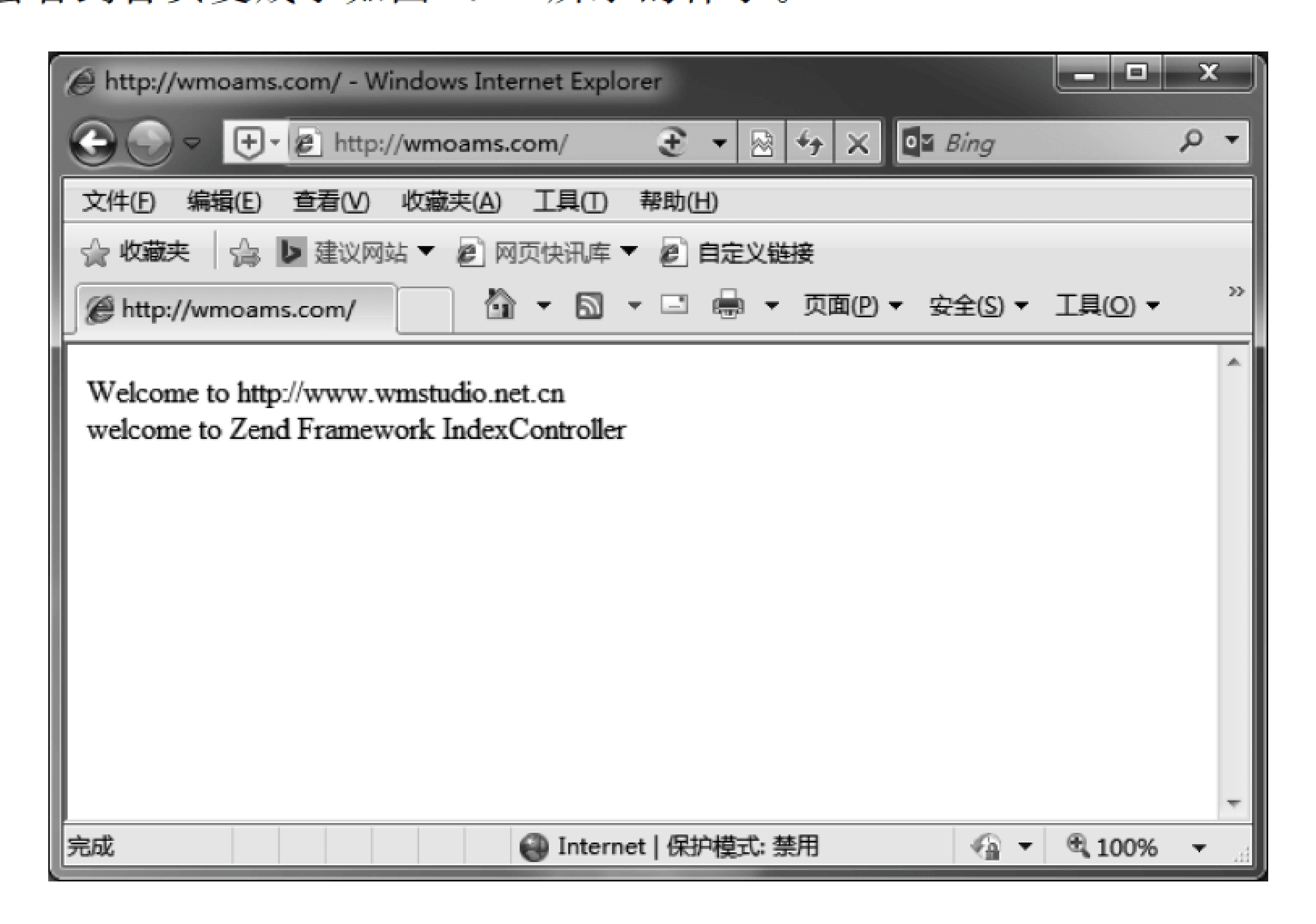


图 2.12　修改控制器方法内容后显示的效果

从图 2.12 所示的页面效果可以看出，在 IndexController 控制器的方法 indexAction 中添加的文本也在页面中有了显示。

由此可以得出结论，如果要修改首页中的显示内容，既可以在视图文件中进行，也可以在控制器的方法中进行。这也说明了项目根目录中的 index.php 文件、Index 控制器、Index 控制器的 index 方法以及视图文件 views\scripts\index\index.phtml 之间必定存在着某种联系。

2.3　Zend Framework 运行原理

如果要弄清楚 2.2 节中 Zend Framework 应用程序的运行机制，必须了解 Zend Framework 的工作原理和流程。要彻底地明白这一点，并不是一件容易的事，对于初学者来说也没有必要一下子全部了解。下面从两个方面简单地介绍一下。

2.3.1　MVC 模式

Zend Framework 框架是一个良好的 MVC 模式的框架，所谓 MVC 模式，其实是一种软件构架模型，即模型(Model)—视图(View)—控制器(Controller)模型，它把软件系统分成 3 个基本部分，即模型、视图和控制器，将数据的处理与显示完全分离。

1. 模型

模型用于封装与程序的业务逻辑相关的数据，以及处理这些数据的方法。它拥有对数据直接访问的权利，包括对数据库的直接访问。“模型”并不关心自身会被如何显示，或是被

如何操作。例如,在本书的案例项目——微梦办公自动化管理系统(WMOAMS)——中,可以把“公文”等数据封装在模型 Wm_Model_File 类的对象里,并在 Wm_Model_File 对象中添加一些对“公文”进行访问的方法,包括添加“公文”、修改“公文”和删除“公文”等。

2. 视图

视图用来显示数据和处理用户交互,把模型数据以特定形式展示给用户。在视图中一般没有程序逻辑,它只是从模型获得显示信息,对于相同的信息可以有多个不同的显示形式或视图。例如在本书案例 WMOAMS 系统中,当 Model 改变时,视图的责任是保持模型与其显示之间的一致性。

3. 控制器

控制器用于控制应用程序的流程,起到不同层面间的组织作用。它处理事件并做出响应,事件包括用户的行为和数据模型上的改变。例如在 WMOAMS 系统中,我们就创建了一个中心路由的 FileController 控制器,用户的所有关于“公文”数据的请求都进入这个控制器,该控制器根据请求的类型来操作 Wm_Model_File“公文”模型。

模型、视图与控制器的分离使应用程序将数据的保存、处理与数据的显示功能分开,降低了组件之间的耦合度。

在经典的 MVC 模式中,事件由控制器处理,控制器再根据事件的类型改变模型或视图,反之亦然。具体地说,每个模型对应着一系列的视图列表,这种对应关系通常采用注册完成,即把多个视图注册到同一个模型。当模型发生改变时,模型向所有注册过的视图发送通知,接下来,视图从对应的模型中获得信息,然后完成视图显示的更新。

相对于早期的 MVC 程序设计思想,Web 环境下的 MVC 模式又有所不同。因为对于一般应用程序而言,可以将视图注册给模型,当模型数据发生改变时,即时通知视图页面做出相应改变;而对于 Web 应用而言,即使将多个视图页面注册给同一个模型,当模型发生变化时,模型也无法主动发送消息给视图页面。因为,Web 应用都是基于请求/响应模式的,只有当用户请求浏览该页面时控制器才负责调用模型数据更新视图页面。

通过上面的分析,可以用图 2.13 表示 Zend Framework 项目中 MVC 架构的工作流程。

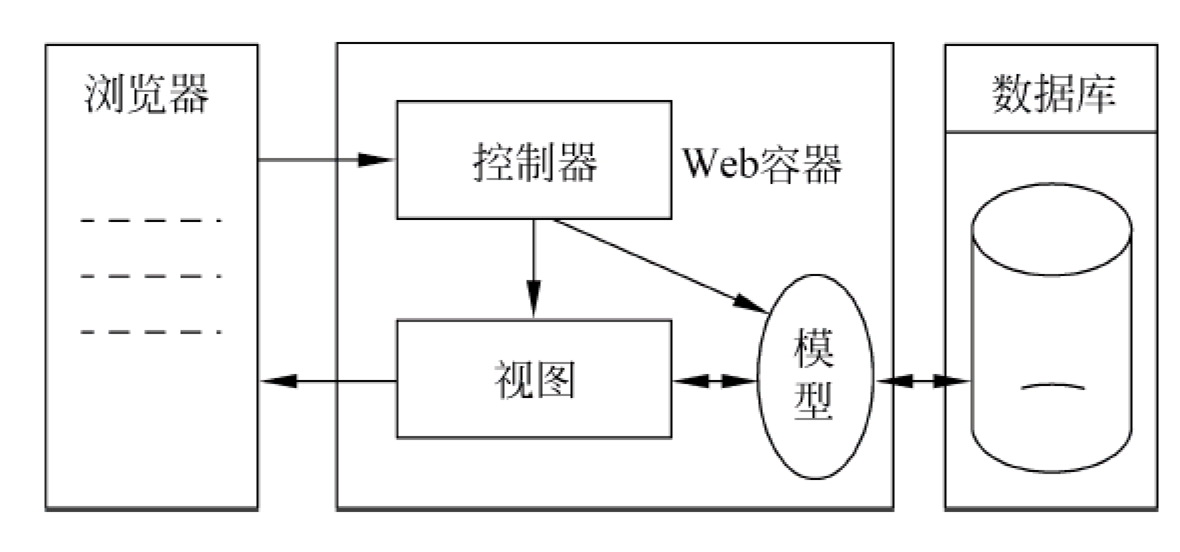


图 2.13　Web 应用中的 MVC 模式流程

在 2.2 节中,通过修改 Index 控制器的 index 方法改变了主页页面的显示内容,说明了视图与控制器的关联关系,这种关系正是图 2.13 描述的。由于文本数据没有存储在数据库中,因此还不能体现出与模型的联系。

这里需要强调的是,MVC 并不是针对某种特定的语言所特有的设计思想,也并不是 Web 应用所特有的模式,它是所有面向对象程序设计语言都应该遵守的规范。

2.3.2 Zend Framework 路由与分发规则

图 2.13 描述的工作流程，只有中间和右边部分对 Zend Framework 是正确的。回顾一下在 2.2 节中所讲的内容，运行项目时输入的 URL 地址是 http://wmoams.com，这里很明显访问的是项目根目录 public 中的 index.php 文件，并没有直接访问控制器。所以，图 2.13 的左边所描述的浏览器与控制器的请求/响应方式对 Zend Framework 应用来说是不正确的。

在 Zend Framework 应用中，浏览器与控制器之间还存在着一个“桥梁”，这个“桥梁”便是前端控制器(Front Controller)。Zend Framework 不仅采用了 MVC 模式，还实现了前端控制模式。这就意味着访问 Zend Framework 应用的所有 HTTP 请求都会被转发到同一个入口，然后再路由到相应的控制器。这个入口便是项目 public 根目录下的 index.php 文件，我们把它称为引导文件。

Zend Framework 通过 URL Rewrite 技术实现前端控制。在 public 根目录下，用户可以看到有一个名为.htaccess 的文件，其内容如下：

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} -s [OR]
RewriteCond %{REQUEST_FILENAME} -l [OR]
RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^.*$ - [NC,L]
RewriteCond %{REQUEST_URI}::$1 ^(/.+)(.+)::\2$
RewriteRule ^(.*)$ - [E=BASE:%1]
RewriteRule ^(.*)$ %{ENV:BASE}index.php [NC,L]
```

以上配置将根目录下没有的文件和路径请求全部转发到了 index.php 上，然后通过 index.php 来路由。在配置文件中有一些正则表达式，如果读者不太理解，请查阅其他 PHP 教程。

上面的流程发生在前端控制器内部，这个过程常常是在前端控制器调用 dispatch()方法时触发的，大致可以分解为以下 12 个步骤：

(1) 产生一个请求 Request，即创建一个 Request Object 对象。

(2) 路由事件 routeStartup 被触发。

(3) 路由器 Router 开始处理这个请求，从中获取请求信息。

(4) 路由事件 routeShutdown 被触发，路由过程结束。

(5) 派遣事件 dispatchLoopStartup 被触发。

(6) 派遣 preDispatch 事件被触发。

(7) 派遣过程中调用动作控制器(action controller)。

(8) 动作控制器将处理完成信息直接写入响应对象(response Object)。

(9) 派遣 postDispatch 事件被触发。

(10) 检测派遣标志，即检查是否还有动作没有完成，如果有再次进入派遣循环。

(11) 派遣事件 dispatchLoopShutdown 被触发。

(12) 产生的响应 Response 被返回。

上面的流程有点复杂，这里不做详细说明。如果大家有兴趣，可以把 Zend Framework

库中的 Zend_Controller_Front 类和 Zend_Controller_Plugin_Broker 类打开研究一下。这两个类文件分别位于 Zend Framework 类库的 Zend\Controller 目录和 Zend\Controller\Plugin 目录下。

2.3.3　Zend Framework 访问流程

通过上面的分析，针对浏览器中的 http://wmoams.com 请求，Zend Framework 应用的大致访问流程是"重写文件.htaccess"→"引导文件 index.php"→"配置文件 application.ini"→"启动类 Bootstrap"→"默认控制器 IndexController"→"视图文件 index.phtml"。

2.4　Zend Framework 文件

上面对 Zend Framework 应用的运行原理进行了简单的分析，大家可能还是不太明白，下面针对项目中的代码再来分析一下 Zend Framework 的访问过程。

1. 引导文件 index.php

引导文件位于项目的 public 目录，其代码如下：

```
<?php
//应用路径
defined('APPLICATION_PATH')
    || define('APPLICATION_PATH',realpath(dirname(__FILE__) . '/../application'));
//应用环境
defined('APPLICATION_ENV')
    || define('APPLICATION_ENV',(getenv('APPLICATION_ENV') ? getenv('APPLICATION_ENV') :
'development'));
//指定工程包含目录
$paths = array(realpath(APPLICATION_PATH . '/../library'));
if (function_exists('zend_deployment_library_path') && zend_deployment_library_path('Zend
Framework 1')) {
        $paths[] = zend_deployment_library_path('Zend Framework 1');
}
$paths[] = get_include_path();
set_include_path(implode(PATH_SEPARATOR, $paths));
//包含 application.php
require_once 'Zend/Application.php';
//实例化 Zend_Application 类
$application = new Zend_Application(
    APPLICATION_ENV,
    APPLICATION_PATH . '/configs/application.ini'
);
$application->bootstrap()->run();     //调用启动类并运行项目
```

在上述代码中，首先定义了两个常量 APPLICATION_PATH 和 APPLICATION_ENV，分别用来保存应用路径和应用环境，相当于两个全局量，它们在项目中的任何地方都可以直接使用。然后将项目的 library 库文件目录和 Zend Framework 类库地址作为数组元素保存在名为 paths 的数组中，并使用 implode()函数将上述路径用 PATH_SEPARATOR 常量连接。最后使用 set_include_path()方法将路径导入到项目中，这样在项目中就可以直接找到

这些路径下的文件了。

Zend Framework 类库被导入到项目中之后，紧接着使用 require_once 文件包含语句包含 Zend 目录下的 Application. php 文件，之后实例化 Zend_Application 类，并通过引用该类的 bootstrap()方法调用启动类，最后通过调用 run()方法运行项目。

2. 项目配置文件 application. ini

在上述 index. php 引导文件的末尾实例化 Zend_Application 类的时候，为类的构造函数传入了一个名为 application. ini 的文件，该文件即为工程的配置文件，默认位于 application\configs 目录下。

打开该配置文件，其内容如下：

```
[production]

phpSettings.display_startup_errors = 0
phpSettings.display_errors = 0
includePaths.library = APPLICATION_PATH "/../library"
bootstrap.path = APPLICATION_PATH "/Bootstrap.php"
bootstrap.class = "Bootstrap"
appnamespace = "Application"
resources.frontController.controllerDirectory = APPLICATION_PATH "/controllers"
resources.frontController.params.displayExceptions = 0

[staging : production]

[testing : production]
phpSettings.display_startup_errors = 1
phpSettings.display_errors = 1

[development : production]
phpSettings.display_startup_errors = 1
phpSettings.display_errors = 1
resources.frontController.params.displayExceptions = 1
```

配置文件的存放位置和名称并不固定，只要与引导文件 index. php 中实例化 Zend_Application 类时传递的参数一致即可。但要注意，配置文件必须存放在用户不能访问到的目录中，因为文件中会有系统的一些重要信息，例如数据库的登录用户名及密码等，以免危及系统安全。

上面 index. php 文件实例化 Zend_Application 类时带入的第 2 个参数为 APPLICATION_PATH . '/configs/application. ini'，APPLICATION_PATH 定义为应用目录，即 application 目录，所以上面的参数为 application\configs\application. ini，显然这就是项目的配置文件的地址。另外，Zend Framework 中的配置文件也可以采用其他类型，例如 Zend_Config 对象或数组，这从如下 Zend_Application 类的构造方法的源代码可以了解到。

```
public function _construct( $environment, $options = null)
    {
         $this->_environment = (string) $environment;
        require_once 'Zend/Loader/Autoloader.php';
```

```
        $this->_autoloader = Zend_Loader_Autoloader::getInstance();
        if (null !== $options) {
            if (is_string($options)) {
                $options = $this->_loadConfig($options);
            } elseif ($options instanceof Zend_Config) {
                $options = $options->toArray();
            } elseif (!is_array($options)) {
                throw new Zend_Application_Exception('Invalid options provided; must be
location of config file,a config object,or an array');
            }
            $this->setOptions($options);
        }
    }
```

注意构造方法中的 options 即为配置文件参数，函数体中的 3 处 if 条件指明了配置文件的可能类型。

配置文件以小节的形式组织，小节名称放置在[ ]中，括号中小节名称旁的冒号表示继承关系。例如[development：production]表示一个 development 小节，它继承了 production 小节的属性配置。也就是说，development 小节同时拥有自己特有的和 production 小节中的所有配置。这里的 production、development 表示应用的不同开发阶段。

可以看出，上面的配置文件有 production、staging、testing、development 4 个小节，分别可为生产、演示、测试、开发 4 种开发环境定义不同的配置。production 小节中定义了一些基本的配置参数，其含义见表 2.2。

表 2.2　application.ini 参数配置说明

配　　置	说　　明
phpSettings.display_startup_errors=0	是否显示 PHP 启动过程中的错误信息，0 为否，1 为是
phpSettings.display_errors=0	是否将错误信息作为输出的一部分显示到屏幕上，0 为否，1 为是
includePaths.library=APPLICATION_PATH "/../library"	声明 library 路径
bootstrap.path=APPLICATION_PATH "/Bootstrap.php"	声明启动文件 Bootstrap.php
bootstrap.class="Bootstrap"	声明启动类名
appnamespace="Application"	声明项目的命名空间为 Application
resources.frontController.controllerDirectory = APPLICATION_PATH "/controllers"	定义前端控制器路径
resources.frontController.params.displayExceptions=0	是否抛出异常错误

staging、development 小节继承了 production 小节的所有属性，并且对其中显示错误和抛出异常的配置进行了改写，以使在测试和开发过程中能得到详细的错误提示和异常抛出信息。

Zend Framework 配置文件中的所有配置实际上是一些全局资源，也可以自定义，然后在项目中的其他地方引用。例如，可以在配置文件的 development 小节的最后添加如下代码定义一个默认域名：

```
default.domain = "http://www.wmstudio.net.cn"
```

配置文件中的配置项在 Zend Framework 中实际上是一个注册表对象的属性,可以通过下面的方式查看。

(1) 在 application\Bootstrap.php 中添加注册对象的初始化方法。

```
class Bootstrap extends Zend_Application_Bootstrap_Bootstrap
{
    protected function _initRegister ()
    {
        //配置文件
        $config = $this->getOptions();        //得到配置文件中的配置项
        Zend_Registry::set('config', $config);  //设置注册表对象
    }
}
```

(2) 在 IndexController 控制器中添加代码,输出配置项与自定义域名。

```
class IndexController extends Zend_Controller_Action
{
    public function init()
    {
        ⋮
    }
    public function indexAction()
    {
            ⋮
            $config = Zend_Registry::get('config'); //得到注册表对象
            echo $config['default']['domain'];      //在视图中显示自定义域名
            echo "<pre>";
            print_r($config);                       //输出注册表对象
            echo "</pre>";
            exit();
    }
}
```

启动 Apache 服务器,在浏览器地址栏中输入 http://wmoams.com,得到如图 2.14 所示的页面。

从图 2.14 显示的结构可以看出,由配置文件生成的注册表对象确实是以数组的形式保存的。所以,可以直接通过 $config['default']['domain']的形式引用配置文件的配置项。图中第一行输出的便是我们自定义的域名文本。

3. 启动文件 Bootstrap.php

在上面的演示中,我们使用了启动文件 Bootstrap.php,它是应用程序的启动类 Bootstrap 的定义文件。从文件内容可以看出,启动类 Bootstrap 继承于 Zend_Application_Bootstrap_Bootstrap 类,该基类已经实现了最基本的启动配置工作,所以在启动类中不需要包含任何代码就可以运行最基本的 Zend Framework 应用程序。

继续跟踪 Zend_Application_Bootstrap_Bootstrap 类的定义,就可以弄清楚 2.3 节中介绍的 Zend Framework 的运行原理。

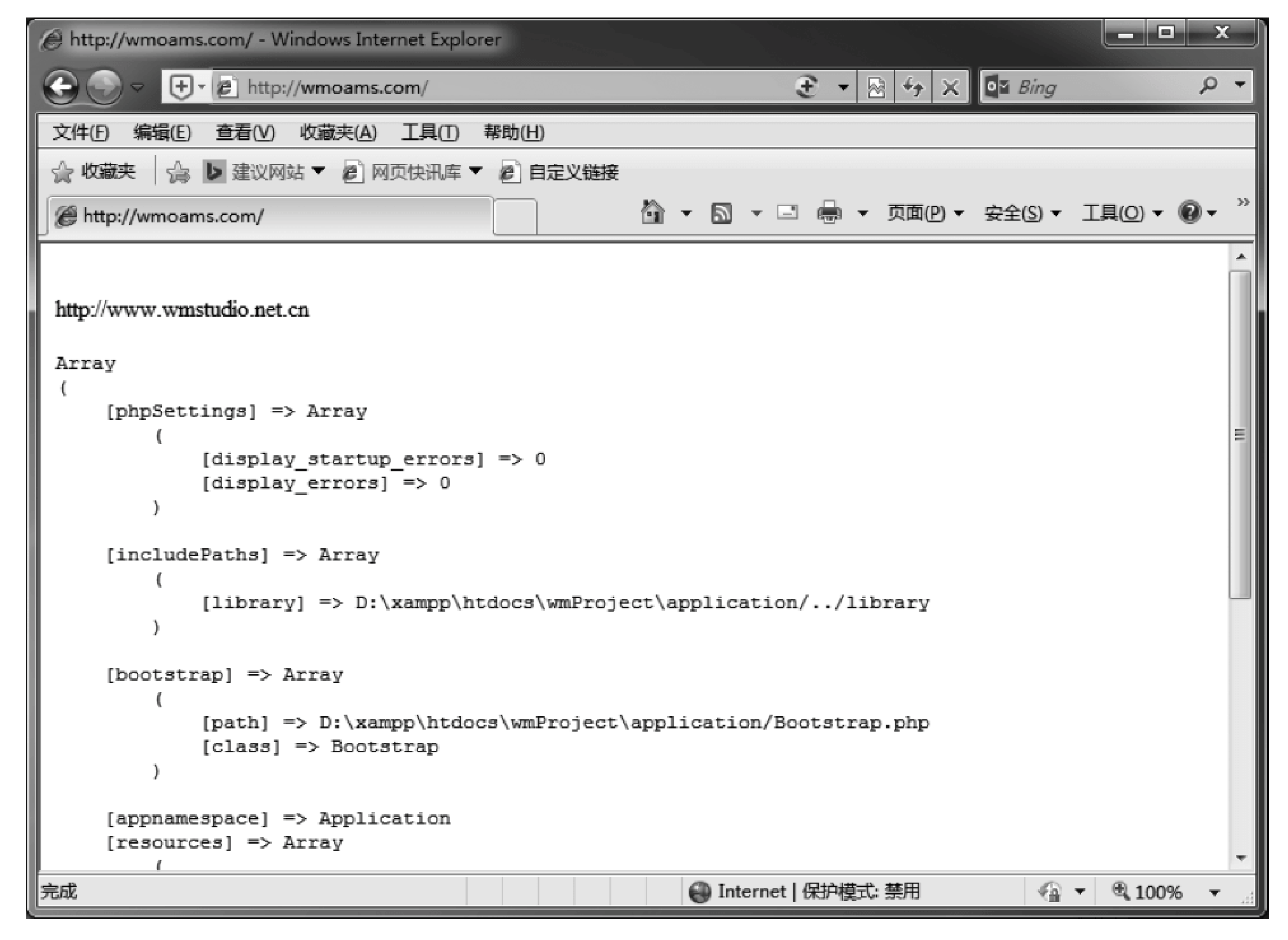


图 2.14　配置对象显示效果

4. 应用程序文件 Application. php

1）动态加载

在分析配置文件 application. ini 时，打开了 Zend_Application 类的构造方法，其中有这样的语句：

```
require_once 'Zend/Loader/Autoloader.php';
```

这里包含的就是 Zend Framework 的动态加载组件的类文件。所谓动态加载，就是把一个 PHP 文件或类通过特殊的语句加载到其他文件中，通过动态加载可以实现 PHP 语句的“一处定义、多处使用”。Zend Framework 中包含大量的文件和类，在使用这些文件和类的时候都是通过动态的方式加载的。

我们知道，PHP 中文件的加载是通过 include()或 require()等函数实现的，在使用这些函数时所带的参数必须是确定的文件名或字符串，而动态加载可以用变量作为参数。下面的代码实现了项目 docs 目录中 readme. txt 文件的加载，注意文件是通过 filename 变量输入参数的。

```
    public function indexAction()
    {
            …
            $filename = APPLICATION_PATH.'/../docs/readme.txt';
           Zend_Loader::loadFile($filename);
    }
}
```

在浏览器地址栏中输入 http://wmoams. com 后的页面效果如图 2.15 所示。



图 2.15　动态加载文件效果

2）类 Zend_Application 方法

Zend_Application 类是应用程序类，用于完成应用程序的一系列初始化工作。这些工作大部分是通过_loadConfig()方法加载配置文件实现的，打开该方法，可以看到如下源代码：

```
protected function _loadConfig( $ file)
    {
        $ environment = $ this - > getEnvironment();
        $ suffix = pathinfo( $ file, PATHINFO_EXTENSION);
        $ suffix = ( $ suffix === 'dist') ? pathinfo(
basename( $ file,". $ suffix"), PATHINFO_EXTENSION)   : $ suffix;
        switch (strtolower( $ suffix)) {
            case 'ini':
                $ config = new Zend_Config_Ini( $ file, $ environment);
                break;
            case 'xml':
                $ config = new Zend_Config_Xml( $ file, $ environment);
                break;
            case 'json':
                $ config = new Zend_Config_Json( $ file, $ environment);
                break;
            case 'yaml':
            case 'yml':
                $ config = new Zend_Config_Yaml( $ file, $ environment);
                break;
            case 'php':
            case 'inc':
                $ config = include $ file;
                if (!is_array( $ config)) {
                    throw new Zend_Application_Exception(
'Invalid configuration file provided; PHP file does not return array value');
                }
```

```
                return $config;
                break;
            default:
                throw new Zend_Application_Exception(
                'Invalid configuration file provided; unknown config type');
        }
        return $config->toArray();
    }
}
```

从代码中的 switch…case 结构的 case 匹配项可以看出，Zend Framework 中的配置文件可以是不同的文件类型，例如 INI、XML、JSON、PHP 等，Zend Framework 会根据不同的类型选用 Zend_Config 组件中的不同对象对它进行读取，并以数组的形式返回。

2.5 本章小结

本章详细介绍了 Zend Framework 应用的创建方法、目录结构、访问流程以及几个重要的框架文件的功能，并对其核心的前端控制技术进行了简要说明。前端控制器的工作原理比较复杂，大家可以先大致了解一下，这样做并不会影响后续的学习。

本章内容是 Zend Framework 应用开发的基础，读者应重点掌握 Zend Framework 框架的 MVC 机制，了解模型、视图和控制器三者的关系，以及它们如何相互配合，从而实现各种业务逻辑和数据显示的协同本质。

第3章 系统概述及总体设计

办公自动化系统(Office Automation System,OAS)是利用技术的手段提高办公效率,进而实现办公自动化处理的信息系统。它采用 Internet/Intranet 技术,基于工作流的概念,使企业内部人员方便、快捷地共享信息,高效地协同工作。OAS 的使用改变了企业过去复杂、低效的手工办公方式,实现迅速、全方位的信息采集、信息处理,从而实现企业管理的信息化与自动化。

从本章开始,我们将以一个办公自动化信息管理系统为例详细讲解 Zend Framework 项目的开发流程,以求在最短的时间内让大家轻松掌握 Zend Framework 的基本结构、规范以及常用组件的使用方法,进而能快速、稳健地开发出满意的 Web 应用程序。

3.1 系统分析

在开发一个项目之前,首先要对所开发的项目进行需求分析、可行性分析,并编制项目计划书,以使项目开发人员了解和掌握系统的前期策划与开发流程。

3.1.1 需求分析

本系统针对某高等学校所属的一所独立学院,其具体情况如下:

(1) 学院是经国家教育部批准设立的实施全日制高等学历教育的普通高等学校,设有 24 个本科专业、17 个专科专业,40 余个专业方向,覆盖文、理、工、农、经、管、法七大学科门类,共有教职工一千余人、各类在校学生 15 000 多人。

(2) 学院经过十多年的发展已经相对稳定,管理也比较规范。原有的办公系统融于教务管理系统中,功能相对简单,现需要将其从原有系统中剥离,并增强其功能。

(3) 学院要求新的 OAMS 主要面向行政办公人员,教师的教务管理不列于其中。

(4) 学院的所有计算机属于局域内网,系统主要在内网运行。学院教职工也可以从学校首页的 OAMS 入口,使用外网进行登录,但不保证网速。

(5) 学院设有学校办公室、人事处、教务处、宣传部、网络中心、后勤服务中心等职能部门,各系部设有系部办公室、学生工作处等部门。学校和各系部都不排除增加其他部门的可能。

(6) 学院管理严格,所有工作必须按计划完成,文件的收发、信息发布的时间、部门、责任人都必须非常清楚。

(7) 学院要求内部文件及信息分级别管理,不同权限的人员浏览不同级别的信息。

(8) 学院要求在新的 OAMS 中能查询到学校主要的新闻及图书馆的最新信息。

3.1.2 可行性分析

可行性分析是目前普遍采用的一种研究工程项目是否可行的科学技术,其通过各种有效的方法对工程项目进行分析,从技术、经济、市场等方面加以评价,最终给投资决策者提供是否选择该项目进行开发的依据。

根据《GB/T 8567—2006 计算机软件文档编制规范》中可行性分析的要求制定本系统项目的可行性研究报告,简要说明如下。

1. 引言

1) 编写目的

为了给企业的决策层提供是否进行项目实施的参考依据,现以文件的形式分析项目的风险、项目需要的投资与效益。

2) 背景

XX 大学 XX 学院是经国家教育部批准设立的实施全日制高等学历教育的普通高等学校。学校为了提高行政管理效率,实现办公自动化,现需要委托其他公司开发一个信息管理系统,项目名称为 XX 办公自动化管理系统。

2. 可行性研究的前提

1) 要求

XX 办公自动化管理系统要求能够提供高等学校日常行政管理的功能,包括公文、事务、消息等的发布、接收、回复、查询,并对各类信息进行审核、删除、检索等管理。同时,要求系统能够提供用户网络空间、日程安排、工作日志等办公功能。

2) 目标

XX 办公自动化管理系统的主要目标是提供方便的日常办公信息发布功能、快捷的信息检索功能以及信息审核、删除等管理功能。

3) 条件、假定和限制

项目需要在两个月内交付用户使用。系统分析师需要在两天内到位,用户需要在 4 天内确认需求分析文档。去除员工两个月的正常休息日 16 天,那么程序开发人员需要用一个月零几天的时间进行系统设计、程序编码、系统测试、程序调试和系统部署工作。

4) 评价尺度

根据用户的要求,系统应以发布、检索功能为主,对于所有信息应能及时准确地保存、审核、查询、备份。由于系统需要在学校各部门运行,系统应具有优良的局域网操作能力,系统中各项操作的延时不能超过 5 秒钟。此外,在系统出现故障时,应能及时进行恢复。

3. 技术可行性分析

XX 办公自动化管理系统的开发采用 Apache、PHP、Zend Framework、phpMyadmin、MySQL 实现,这些开发软件都是免费的,可以直接从网上下载,无须支付任何费用。该系统的业务逻辑相对比较简单,并且公司有类似项目的开发经验,技术上不存在任何阻碍。

4. 投资及效益分析

1) 支出

根据系统的规模及两个月的项目开发周期,公司决定投入 5 个人。因此,公司将直接支付 X 万元的工资及各种福利待遇。在项目安装及调试阶段,用户培训、员工出差等费用支出需要

X 万元。在项目维护阶段预计需要投入 X 万元的资金，累计项目投入需要 XX 万元的资金。

2）收益

用户提供项目资金 XX 万元。对于项目运行后进行的改动，采取协商的原则根据改动规模额外提供资金。因此，从投资与收益的效益比上公司可以获得 XX 万元的利润。

项目完成后，将给公司提供资源储备，包括技术、经验的积累，以后再开发类似的项目时可以极大地缩短项目开发周期。

5. 结论

根据上面的分析，技术上不会存在问题，因此项目延期的可能性很小。在效益上，公司投入 5 个人、两个月的时间获利 XX 万元，比较可观。在公司今后的发展上可以储备办公自动化系统开发的经验和资源，因此认为该项目可以开发。

3.1.3 编写项目计划书

根据《GB/T 8567—2006 计算机软件文档编制规范》中的项目开发计划要求制定本系统项目设计的项目计划书，简要说明如下。

1. 引言

1）编写目的

为了保证项目开发人员按时、保质地完成预订目标，更好地了解项目实际情况，按照合理的顺序开展工作，现以书面的形式将项目开发周期中的项目任务范围、项目团队组织结构、团队成员的工作责任、团队内外沟通的协作方式、开发进度、检查项目工作等内容描述出来，作为项目相关人员之间的共识和约定以及项目周期内的所有项目活动的行动基础。

2）背景

XX 办公自动化管理系统是本公司与 XX 大学 XX 学院签订的待开发 Web 项目，系统性质为日常行政管理信息服务类型，可为学校各部门及全体教职员工提供日常办公信息的发布与检索，是一个综合信息的管理与互动交流平台，项目周期为两个月。

2. 概述

1）项目目标

项目目标应当符合 SMART 原则（目标管理原则），把项目要完成的工作用清晰的语言描述出来。XX 办公自动化管理系统的项目目标如下：

XX 办公自动化管理系统主要为用户提供行政管理信息服务，日常办公中的所有事务都应该包括在内，例如会议通知、公示公告、文件规定、新闻资讯、申请报告、工作安排等。项目运行后，要实现能够为用户的工作带来极大方便并提高学校行政管理效率的目标，整个项目需要在两个月的期限结束后交给客户验收并试运行。

2）产品目标与范围

一方面，XX 办公自动化管理系统能够实现学校行政管理的电子化、无纸化，为学校节省大量的办公及人力资源，降低管理成本；另一方面，XX 办公自动化管理系统为学校提供公共信息交互平台，能大大提高信息传递的快速性、准确性、及时性，提高管理效率。

3）应交付成果

项目开发完成后，应交付的成果内容如下：

（1）以光盘的形式交付 XX 办公自动化管理系统的源程序、系统数据库文件、系统使用

说明书。

(2) 客户方拥有自己的服务器及官方主页,需要乙方协助甲方在其主页的适当位置添加“办公系统”登录接口。

(3) 系统发布到校园网上以后,提供为期6个月的无偿维护与服务,超过6个月后进行系统有偿维护与服务。

(4) 项目验收方式与依据

项目验收分为内部验收和外部验收两种方式。在项目开发完成后,首先进行内部验收,由系统测试员根据用户需求和项目目标进行验收。项目在通过内部验收后交给用户进行验收,验收的主要依据为项目需求说明书及相关规范。

3. 项目团队组织

1) 组织结构

为了完成XX办公自动化管理系统项目的开发,公司组建了一个临时的项目团队,由项目经理、系统分析师、PHP开发工程师、网页设计师和系统测试员组成。

2) 人员分工

为了明确项目团队中每个人的任务分工,现制定人员分工表,如表3.1所示。

表3.1 项目开发人员分工表

姓名	技术水平	所属部门	角色	工作描述
王一浙	计算机应用技术博士、高级工程师	项目开发部	项目经理	负责项目的审批、决策的实施、项目的前期分析和策划、项目开发进度的跟踪、项目质量的检查
李晓峰	高级系统分析师	项目开发部	系统分析师	负责系统功能分析、系统框架设计
郑加清	高级工程师	项目开发部	PHP开发工程师	负责软件前、后台设计与编码
王秀芳	高级美工设计师	设计部	网页设计师	负责网页风格的确定、网页图片的设计
顾向荣	高级系统测试工程师	项目开发部	系统测试员	对软件进行测试、编写软件测试文档

3.2 系统设计

3.2.1 系统目标

本系统是针对高等学校内部自动化管理的要求设计的,主要实现如下目标:

- 键盘操作,快速响应;
- 实现文件类信息的强大管理能力;
- 实现对教职工基础信息的管理功能;
- 实现个人办公的信息自动化管理功能;
- 实现个人办公的即时信息收发功能;
- 实现个人办公的用户网络空间功能;
- 对系统用户进行管理,包括访问权限控制;
- 实现数据备份、系统日志功能;

- 系统最大限度地实现易安装性、易维护性和易操作性；
- 系统运行稳定、安全可靠。

3.2.2 系统功能结构

下面根据系统分析及系统目标给出系统的前、后台管理功能结构图，如图 3.1 和图 3.2 所示。

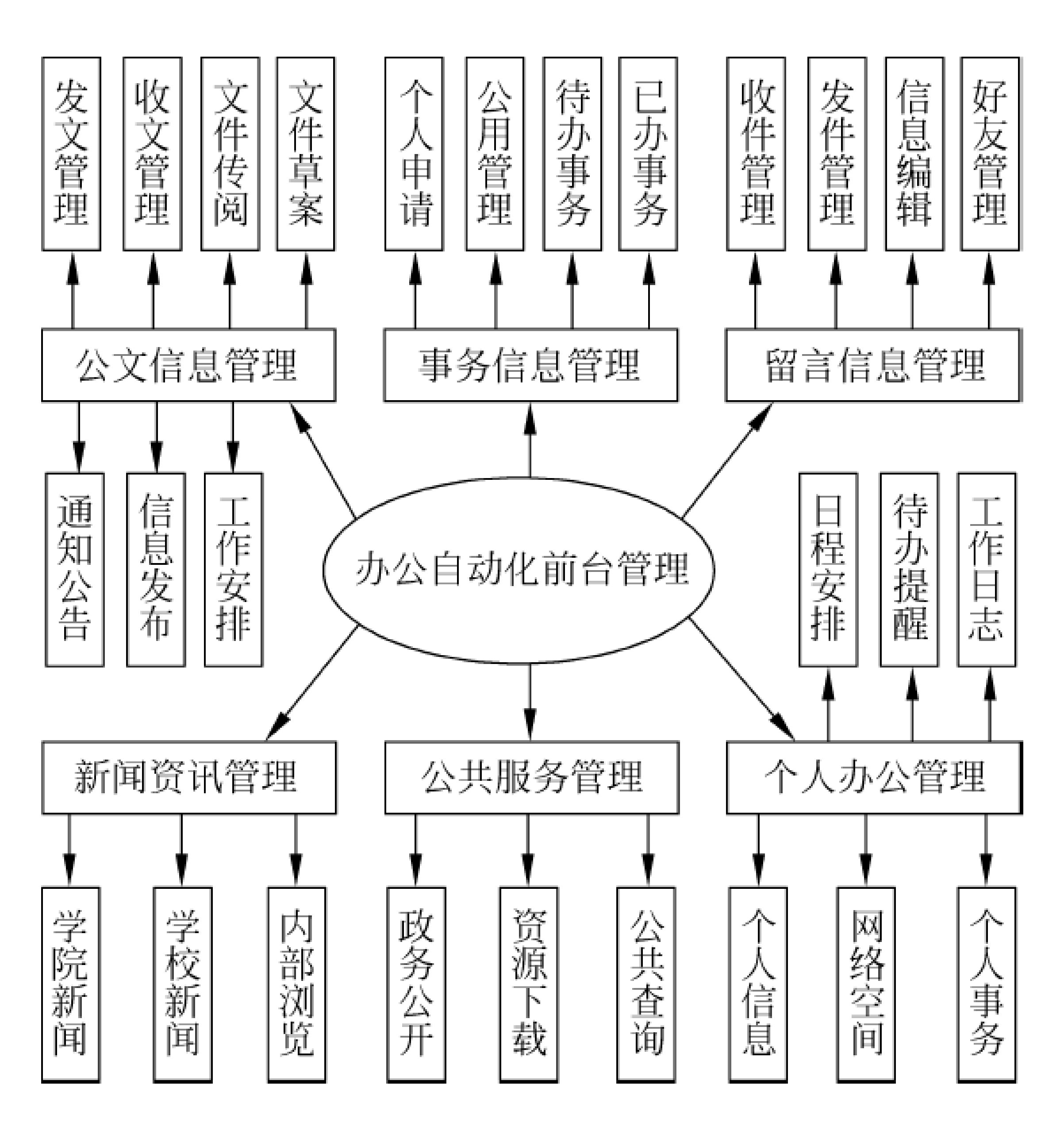


图 3.1　前台管理的功能结构

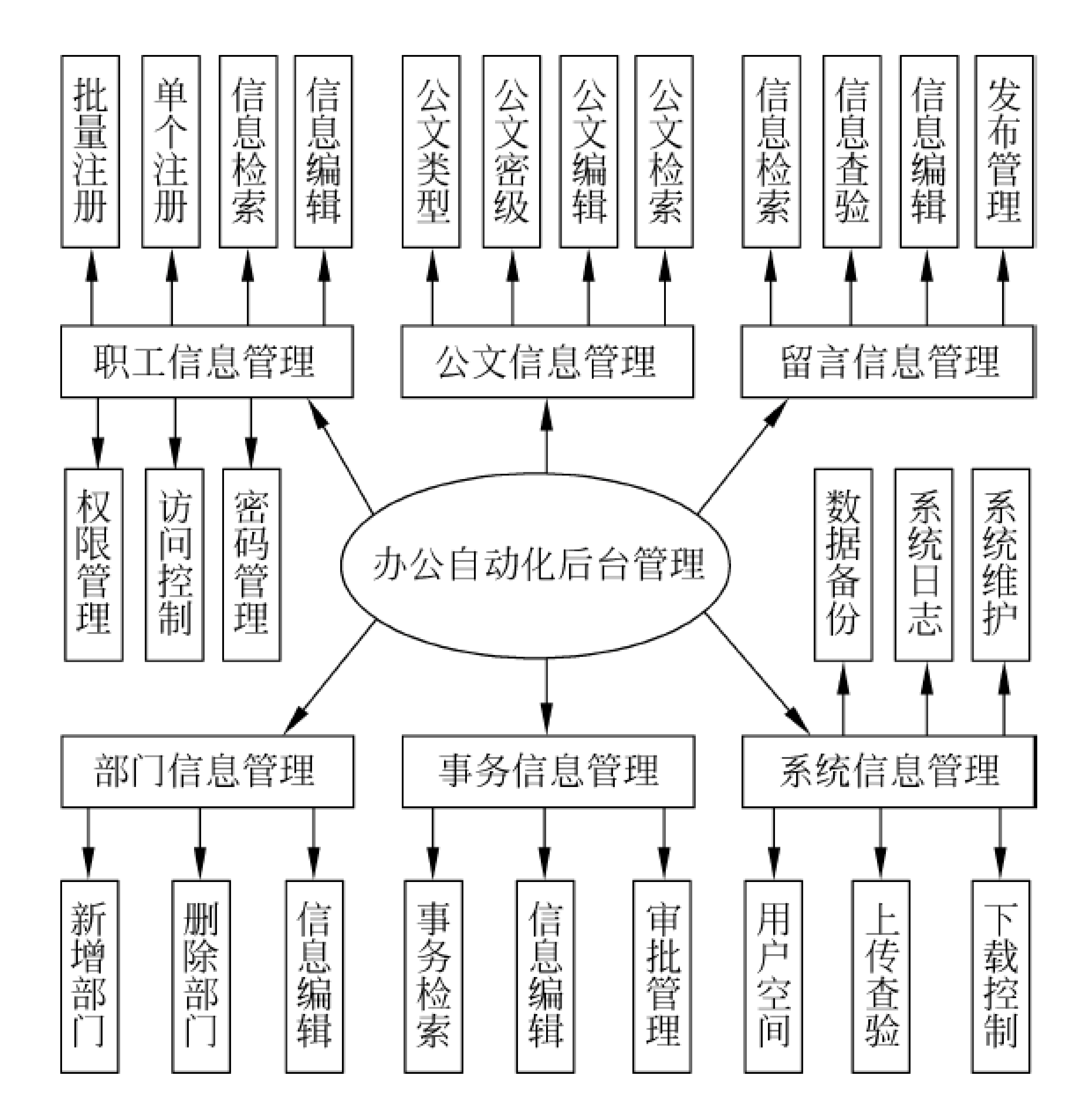


图 3.2　后台管理的功能结构

3.2.3 系统功能预览

系统前、后台管理功能分别由 Zend Framework 的默认模块及自定义 admin 模块实现，图 3.1 和图 3.2 中的各个功能单元分别对应模块中的不同控制器，而功能单元中的每项具体功能，则由控制器的方法完成。

下面仅列出几个典型页面，其他页面请参考源代码或后续章节。

1. 前台登录页面

图 3.3 所示为系统前台登录页面，该页面用于实现对用户登录的信息验证。由于本系统为学校内部使用系统，用户必须通过登录认证后才能进入系统内部页面。用户注册由系统管理员统一操作完成，不允许用户自行注册。为方便用户找回密码，在页面中设置了“忘记密码”超级链接。



图 3.3 前台登录

2. 前台主页

图 3.4 所示为系统前台主页页面，该页面上显示了导航菜单，登录用户的姓名、所属部门及角色等信息，用户可以单击姓名进入个人信息页面对个人信息进行修改。除此之外，页面上还设置了“通知公告”、“最新公文”、“个人日程”、“最新消息”、“待办提醒”以及“工作安排”等重要信息的显示区域，方便用户快速、准确地了解需要处理的工作事务。

3. 公文管理

图 3.5 所示为系统公文信息管理模块主页面，该页面按公文类型分类显示与用户及用户所在部门有关的文件信息，用户单击文件标题即可进入详细内容页面。

4. 留言管理

图 3.6 所示为系统留言信息管理模块主页面，系统留言信息模块是用户个人消息收发平台，有利于用户工作时的及时沟通。该界面模仿了常见的邮件系统。



图 3.4 系统主页



图 3.5 系统公文信息管理

5. 事务管理

图 3.7 所示为系统事务信息管理模块主页面，该模块用于事务的添加以及分类显示，界面借用了图 3.6 的样式。

6. 日程管理

图 3.8 所示为用户日程信息管理模块页面，该模块用于用户日程事务的添加以及分类显示，设置了“今日工作”、“本周安排”、“日程查询”与“待办提醒”等功能项。图中显示的是用户个人日程查询界面，采用了电子台历的方式，符合用户的办公习惯。



图 3.6　系统留言信息管理



图 3.7　系统事务信息管理

7. 网络空间

图 3.9 所示为用户网络空间管理模块页面，系统为用户提供了一定容量的网络存储空间，以方便用户日常办公的需要。在这里用户可以建立自己的文件夹，还可以上传、下载移动文件，它是一个功能齐全的资源管理器。

8. 管理中心

图 3.10 所示为系统后台管理模块页面效果，系统后台管理中心根据用户角色或权限进行访问控制，不允许普通用户进入，部门管理员也只拥有部分页面的访问权限。



图 3.8　个人日程查询页面



图 3.9　用户网络空间管理页面

3.2.4　系统工作流程

用户登录系统后会进行一系列操作，把这些操作的过程和结果以图形的形式表现出来，不仅可以让开发者快速地理清思路，及时解决出现的问题，还可以让使用者很快明白该系统的操作方式与方法。下面给出本办公自动化管理系统的工作流程，如图 3.11 所示。



图 3.10　系统后台管理中心

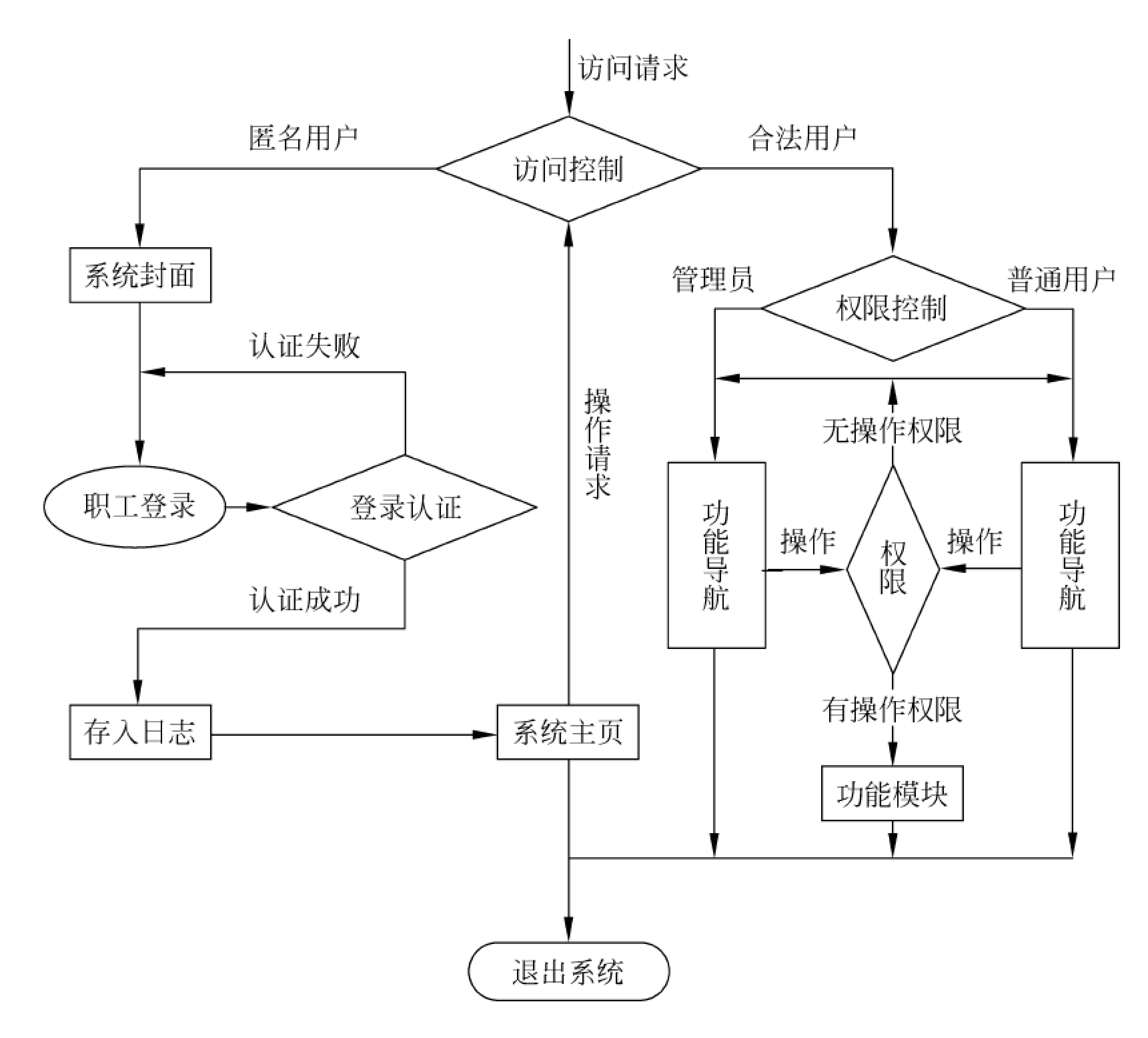


图 3.11　系统工作流程图

3.2.5 开发环境

本系统开发软件环境如下：

- 操作系统：Windows 7 旗舰版；
- 服务器：Apache 2.0；
- PHP 软件：5.5.11；
- 数据库：MySQL 5.6.16；
- MySQL 图形化管理软件：phpMyAdmin 4.1.12；
- 开发工具：Zend Studio 10.6.1、Notepad++6.6.7；
- 浏览器：IE 8.0；
- 分辨率：最佳效果 1366×768。

3.3 数据库设计

系统开发离不开数据库的支持，只有拥有了强大的数据库才能存储大量的数据信息，实现更多、更好的功能为用户服务。本书案例项目是一个办公自动化管理系统，采用了数据表对办公信息进行存储。下面对系统数据库设计做一个简单介绍。

3.3.1 数据库分析

本系统是一个中小型办公信息管理平台，考虑到开发成本、搭配的合理性以及操作的灵活性等，选用 MySQL 数据库。

MySQL 数据库是目前最流行的开源数据库，是完全网络化、跨平台的关系型数据库系统，它是完全免费的，可以在网上任意下载。同时，从匹配的角度来说，PHP 与 MySQL 数据库一直都是公认的最佳搭档。另外，MySQL 数据库不仅可以在命令模式下进行操作，还配备了一些图形化管理工具，例如 phpMyAdmin 等，大大方便了对数据库的操作。

3.3.2 数据库概念设计

根据上述项目需求分析及系统功能将系统实体关系归纳为以下几种。

1. 用户信息实体

用户信息实体包括职工姓名、登录名、密码、性别、年龄、所属部门、角色等表示个人身份的所有人事数据信息及系统使用信息。为了更清晰地表示存储数据的层次关系，实体中还包含了一些依附数据的子实体，例如部门实体等。

2. 公文信息实体

公文信息实体包括公文类型、发布人、发布时间、发布部门、接收单位、处理状态、密级等相关信息，其中还包含公文类型等子实体。

3. 留言信息实体

留言信息实体包括留言主题、留言内容、信息发布人、信息接收人、信息发布与接收时间等，包含用户好友子实体。

4. 事务信息实体

事务信息实体包括申请事务信息子实体与日程安排信息子实体，分别存储事务主题、事务内容、事务处理时间以及事务处理状态等相关信息。

5. 网络空间信息实体

网络空间信息实体包括用户网络空间名称、容量大小、文件名、子目录名等相关信息。

3.3.3 数据库物理结构设计

根据上述数据库概念设计，创建一个名为 db_wmoams 的数据库，其中包含 13 张数据表，如图 3.12 所示。

服务器: 127.0.0.1 » 数据库: db_wmoams

结构 SQL 搜索 查询 导出 导入 操作

表	行数	类型	排序规则	大小	多余
tb_affair	7	MyISAM	gb2312_chinese_ci	2.6 KB	-
tb_depart	17	MyISAM	utf8_general_ci	3.7 KB	-
tb_docs	23	MyISAM	utf8_general_ci	6.8 KB	1.2KB
tb_doctype	15	MyISAM	utf8_general_ci	2.3 KB	-
tb_friend	2	MyISAM	utf8_general_ci	2.1 KB	-
tb_news	11	MyISAM	utf8_general_ci	4.9 KB	-
tb_receive	~5	InnoDB	utf8_general_ci	16 KB	-
tb_reply	1	MyISAM	gb2312_chinese_ci	2.1 KB	-
tb_schedule	2	MyISAM	gbk_chinese_ci	2.1 KB	-
tb_send	~7	InnoDB	utf8_general_ci	16 KB	-
tb_userfile	3	MyISAM	gbk_chinese_ci	4.2 KB	52字节
tb_userfolder	5	MyISAM	gbk_chinese_ci	4.2 KB	-
tb_users	46	MyISAM	utf8_general_ci	9.3 KB	-
13 张表 总计	144	InnoDB	latin1_swedish_ci	76.4 KB	1.3 KB

图 3.12 系统数据库中的数据表

图 3.12 中的数据表从上至下依次为事务表、部门表、公文表、公文类型表、好友表、新闻资讯表、留言收件表、事务批示表、日程表、留言发件表、用户文件表、用户目录表以及用户信息表。为了节省篇幅，将各数据表的详细设计移至后续章节中，这里暂不介绍。

3.4 公共文件设计

公共文件就是将系统中可能在不同的地方被使用多次的代码编辑而成的单独文件，在使用时只需要用 include 或 require 等语句将文件包含进来即可。这些文件一般包括数据库的连接、分页管理、模板配置、自定义类、CSS 样式以及 JS 脚本等文件，依系统设计模式变化。

本系统采用 Zend Framework 框架技术，与页面视图有关的公共代码将由 Zend Framework 提供的“助手”技术实现，而公共的业务逻辑部分由自定义的公共控制器管理。所以，本案例系统中的公共文件主要指页面的 CSS 样式文件以及 JS 脚本文件，对于这部分内容请读者参考本书源码。

3.5 本章小结

本章针对案例项目简要介绍了 Web 应用开发的一般步骤，包括系统分析、系统设计等，旨在为大家呈现一个项目开发的完整过程。软件作为一种产品，有其特有的生产管理方法以及应遵循的标准规范，这些都是大家必须要了解的。

本章内容比较简单，并且不同的企业、不同的项目类型还会存在很大的差异，学习本章重点掌握所介绍的项目开发及管理流程。

第 4 章 页面设计及 layout 布局模板

对于一个办公自动化管理系统来说,其主要作用是帮助企业提高管理效率和办公效率,因此系统页面不可能、也不允许进行复杂的设计,每个页面上显示的内容也不能太多,应做到简洁大方、重点突出、导航方便。

本章通过系统首页、用户登录以及系统主页页面的设计介绍系统页面布局、CSS 样式表等常用页面设计方法,重点介绍 Zend Framework 中的 layout 布局模板。

4.1 系统初始设置

通过第 2 章的分析,读者基本了解了 Zend Framework 应用的项目结构及运行机制。除了配置文件中的基本设置外,在系统开发过程中还需要添加一些自己的配置,例如页面的共有设置、注册表的设置、缓存的设置、认证的设置等。

4.1.1 页面共有属性设置

1. 视图对象初始化

系统的初始化一般在应用程序的启动类 Bootstrap 中完成。启动类 Bootstrap 中的初始化方法均以_init 为前缀,它们不是 PHP 内置的,而是由 Zend Framework 框架开发者自己定义的。框架会在执行其他程序之前执行这些方法,这实际上和 PHP 内置的构造函数_construct 类似,不同之处在于构造函数是 PHP 内置的,由 PHP 解析引擎自动调用,它处于语言级;而初始化方法是由 PHP 框架自动调用的,处于框架级,因此后者比前者使用起来更加灵活。

打开启动文件 application\Bootstrap.php,在类 Bootstrap 中添加一个名为_initView 的初始化方法,并实例化一个视图对象,代码如下:

```
class Bootstrap extends Zend_Application_Bootstrap_Bootstrap
{
  protected function _initView(){
      $ view = new Zend_View();
      return $ view;
  }
}
```

上述代码中的类 Zend_View 是 Zend Framework 用来处理 MVC 模式中视图功能的类,使用它可以将视图部分的代码与模型及控制器部分的代码进行分离。这里直接采用

new 方法创建视图对象，这个视图对象将成为我们项目的视图。大家可能会觉得奇怪，在第 2 章中，我们已经成功运行了项目，但那时并没有定义视图对象，为什么也能显示视图页面呢？其实，在 Zend Framework 框架的每个控制器中都有一个 initView 方法，通过这个方法，控制器能获得已经存在的视图对象，一旦检测到视图对象不存在，则会创建它。该初始化方法是 Zend_Controller_Action 类的一个成员函数，大家可以打开 Zend Framework 库中的 Zend\Controller\Action.php 文件，看一下里面的源代码。

2. 配置页面资源

在配置文件 application\configs\application.ini 的 development 小节添加如下配置代码，设置系统页面的一些共有属性。

```
pageInfo.default.doctype = "XHTML1_STRICT"
pageInfo.default.title = "微梦办公自动化管理系统"
```

上述代码的第 1 句设置系统页面的文档类型；第 2 句设置系统默认模块的视图页面标题。视图页面的文档类型主要包括 XHTML11、XHTML1_STRICT、XHTML1_ TRANSITIONAL、XHTML1_FRAMESET、XHTML1_BASIC1、HTML4_ STRICT、HTML4_LOOSE、HTML4_ FRAMESET 等。

3. 添加页面类型

本系统视图页面采用 XHML 格式，在视图对象中添加页面类型 XHTML_STRICT。

```
class Bootstrap extends Zend_Application_Bootstrap_Bootstrap
{
protected function _initView(){
        $pageInfo = $this->getOption('pageInfo');
         $view = new Zend_View();
         $view->doctype($pageInfo['default']['doctype']);
        return $view;
    }
}
```

首先获取配置文件中的 pageInfo 资源，然后以参数的方式将 doctype 的值传入视图对象中。设置 HTML 页面中的文档类型(DOCTYPE)属性为 XHTML1_STRICT，即相当于在页面中添加了如下标签内容。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

4. 添加页面标题

本系统的所有视图页面标题都设置为“微梦办公自动化管理系统”，通过参数的方式向视图中添加。

```
class Bootstrap extends Zend_Application_Bootstrap_Bootstrap
{
  protected function _initView(){
        $pageInfo = $this->getOption('pageInfo');
         $view = new Zend_View();
        $view->doctype($pageInfo['default']['doctype']);
```

```
        $view->headTitle($pageInfo['default']['title']);
        return $view;
    }
}
```

上面的阴影部分代码设置视图页面中的标题(title)值为“微梦办公自动化管理系统”,即相当于在页面中添加了如下标签内容:

```
<title>微梦办公自动化管理系统</title>
```

为了能够尽可能多地介绍 Zend Framework 的组件功能,这里采用了通过配置文件设置视图页面属性的方法。上述代码中使用的 getOption 方法是 Zend_Application_Bootstrap_Bootstrap 类的基类 Zend_Application_Bootstrap_BootstrapAbstract 的成员函数;而代码中的 doctype 和 headTitle 看起来好像是 Zend_View 类的成员函数,但在视图类 Zend_View 及其基类 Zend_View_Abstract 中却找不到它们。其实,它们是 Zend Framework 的视图助手,关于“助手”的概念,我们将在第 11、12 章中详细介绍,大家暂且把它们当作方法使用即可。

__注意__,这里只是在视图对象中设置了“文档类型”与“页面标题”,如果要在视图页面中使用它们,还必须在页面中用 PHP 的 echo 进行输出。

4.1.2 对象注册表设置

1. 添加注册初始化方法

为了在系统的其他地方使用 application.ini 中的自定义配置项,在 Bootstrap 类中再添加一个_initRegister 方法,代码如下:

```
class Bootstrap extends Zend_Application_Bootstrap_Bootstrap
{
  …
  protected function _initRegister() {

  }
}
```

2. 创建对象注册表

在_initRegister 方法中实例化一个 Zend_Registry 类对象,并将配置文件 application.ini 中的所有配置项放入注册表对象中。

```
class Bootstrap extends Zend_Application_Bootstrap_Bootstrap
{
  …
  protected function _initRegister()
  {
     $config = $this->getOptions();
     Zend_Registry::set('config', $config);
  }
}
```

代码中的 Zend_Registry 是 Zend Framework 框架的对象注册表类,它是应用程序中值

与对象的存储容器。将程序中的值或对象存储在注册表中,就可以在程序中随时随地引用,也可以把它简单地理解为一种特殊的全局变量。

上述代码中的 set 方法是 Zend_Registry 类的静态方法,这里使用这个方法为注册表对象设置内容,即将系统配置文件中的配置项全部放入到注册表对象中。注意,采用 set 方法设置的注册表内容在使用时需要用 get 方法获取。当然,也可以采用传统的 new 方法实例化一个注册表对象,代码如下:

```
$ registry = new Zend_Registry(array('config' => $ config)
```

这里返回的值 registry 就是一个对象注册表实例。

4.1.3 会话设置

在计算机语言中,会话是一种面向连接的可靠通信方式。在 PHP 中,它代表服务器端与客户端之间的一种持久的状态数据。使用 Zend Framework 的 Zend_Session 组件可以在由相同客户端发起的多个页面请求之间管理和保护会话数据。

为了后续项目开发的需要,这里先创建一个名为 session 的会话空间。打开 Bootstrap.php 启动文件,继续添加代码:

```
class Bootstrap extends Zend_Application_Bootstrap_Bootstrap
{
  …
  protected function _initRegister ()
  {
      …
      $ session = new Zend_Session_Namespace('session');
      Zend_Registry::set('session', $ session);
  }
}
```

4.1.4 缓存设置

缓存是 Web 应用中几乎都要用到的性能优化技术方案,使用缓存可以极大地减轻服务器的负载。关于缓存的原理及使用,我们将在第 12 章中进行详细介绍。

首先在配置文件 application\configs\application.ini 中设置缓存参数,包括失效时间与存储路径。打开配置文件,在 development 小节中添加如下代码:

```
cache.leftTime = "3600"
cache.cache_dir = APPLICATION_PATH "/cache/"
```

这里设置缓存的失效时间为两个小时,缓存数据存储目录为应用目录下的 cache 子目录。该子目录需要用户自己手工创建。

缓存的参数有很多,这里只简单地配置了两项。准备好缓存参数后,就可以在初始化方法中创建缓存对象了。代码如下:

```
class Bootstrap extends Zend_Application_Bootstrap_Bootstrap
{
```

```
    …
  protected function _initRegister ()
  {
        …
        $ frontOptions = array('leftTime' => $ config['cache']['leftTime'],
                                'automatic_serialization' => true);
        $ backOptions = array(
                          'cache_dir' => $ config['cache']['cache_dir']);
        $ cache = Zend_Cache::factory('Core', 'File', $ frontOptions, $ backOptions);
        Zend_Registry::set('cache', $ cache);
    }
}
```

4.1.5 认证对象设置

Zend Framework 提供了用户访问认证功能，该功能由 Zend_Auth 组件实现。用户的认证信息影响了该用户在系统中的访问权限。由于在应用的很多地方都要使用认证信息，所以，我们把它作为一个全局对象来处理。

在启动文件中继续添加代码：

```
class Bootstrap extends Zend_Application_Bootstrap_Bootstrap
{
    …
protected function _initRegister ()
    {
        …
        $ auth = Zend_Auth::getInstance();
        $ storage = new Zend_Auth_Storage_Session('session', 'user');
        $ auth -> setStorage( $ storage);
    }
}
```

在上述代码中，首先使用 Zend Framework 的认证组件 Zend_Auth 类的静态方法获取认证对象，然后创建认证对象的信息存储空间。

4.2 CSS 样式表

本书主要讲解 Zend Framework 框架，重点介绍 Zend Framework 中组件的使用方法，因此，CSS 样式文件并不属于本书讲解的范围，但作为一个 Web 应用，这项内容又是必不可少的，而且在实际开发过程中没有专职美工的情况下，常常会花费大量的时间与精力。

为了让注意力集中于框架编程本身，将本书案例项目的所有 CSS 样式文件存放在学习辅导网站(微梦网)上，请大家到网站相关目录下载，网址为 http://www.wmstudio.net.cn。

在后续的项目开发过程中，为了保持项目 MVC 模式中各部分的完整，并让大家在学习过程中能及时看到页面效果，我们对各控制器的视图代码做了说明，但大多非常简单。为了帮助大家更好地理解后续的页面布局，下面对 CSS 样式及相关背景做一个简单介绍。

4.2.1 Web 标准布局

虽然传统的网页布局(即使用 table 的布局)已经有很长的历史和较成熟的技术规范了,但其存在的缺陷也是非常明显的。在这种布局中,由于页面的内容和修饰没有分离、页面代码的语义不明确,导致页面升级与数据利用都非常困难。

建立 Web 标准就是要解决网站中由于浏览器升级、网站代码臃肿、代码不易用等带来的诸多问题,它由 W3C(W3C.org 万维网联盟)组织与实施。

Web 标准分为 3 个方面,即结构标准语言、表现标准语言和行为标准。

1. 结构标准语言

结构标准语言包括 XML 和 XHTML 两个部分。XML 是 The Extensible Markup Language 的简写,它是一种扩展式标识语言。XML 设计的目的是对 HTML 补充,它具有强大的扩展性,可以用于网络数据的转换和描述。同时 XML 具有简洁有效、易学易用、开放的国际化标准、高效可扩充等特点。XHTML 是 The Extensible HyperText Markup Language(可扩展标识语言)的缩写,XHML 是基于 XML 的标识语言,是在 HTML 4.01 的基础上用 XML 的规则对其进行扩展建立起来的,它是 HTML 向 XML 的过渡。

2. 表现标准语言

表现标准语言包括 CSS,它是 Cascading Style Sheets(层叠样式表)的缩写。目前推荐遵循的是 W3C 于 1998 年 5 月 12 日推出的 CSS2。CSS 标准建立的目的是以 CSS 进行网页布局,控制网页的表现。CSS 标准布局与 XHTML 结构语言相结合,可以实现表现与结构相分离,提高应用的使用性和可维护性。

3. 行为标准

行为标准也包括两个部分,即 DOM 和 ECMAScript。DOM 是 Document Object Model(文档对象模型)的缩写。W3C 建立的 W3C DOM 是建立网页与 Script 或程序语言沟通的桥梁,它实现了访问页面中标准组件的一种标准方法。ECMAScript 是 ECMA(European Computer Manufacturers Association)制定的标准脚本语言。

4.2.2 Web 标准的优势

使用 Web 标准可以大大缩减页面代码,提高浏览速度,缩减带宽成本。由于其结构清晰,也能使网页更容易被搜索引擎搜索。因此,它对 Web 应用的浏览者及拥有者都是有益的。

1. 对于 Web 应用的浏览者

- 页面代码量减少,文件下载速度更快,同时浏览器显示页面的速度更快;
- 清晰的语义结构,使得内容能被更多的用户所访问;
- 实现了结构和表现相分离,内容能被更多的设备(包括手机、打印机等)所访问;
- 由于样式文件的独立性,用户选择自己喜欢的界面变得更容易;
- 由于可以调用独立的打印样式文件,所以便于页面的打印。

2. 对于 Web 应用的拥有者

- 代码变得更简洁、组件用得更少,使维护变得很容易;
- 对带宽要求降低,节约了成本;

- 页面结构的语义性清晰，使搜索引擎的搜索变得更容易；
- 实现了结构和表现的分离，使得在对页面外观进行修改的时候不改变页面内容；
- 可以调用不同的样式文件，使得提供打印版本变得更容易；
- 清晰合理的页面结构，提高了 Web 应用的易用性。

4.2.3 CSS 样式基础

层叠式样式表 CSS 是 W3C 组织制定的用于控制网页样式的一种标记语言，它包括 CSS1 和 CSS2 两部分。其中，CSS2 是于 1998 年 5 月发布的，包含了 CSS1 的内容，它也是现在通用的标准。

1. CSS 语法

通常情况下，CSS 的语法包括 3 个方面，即选择符、属性和值。其格式如下：

```
选择符{属性: 属性值; }
```

说明：

- 属性必须包含在{ }号之中；
- 属性和属性值之间用"："分隔；
- 当有多个属性时，用"；"进行区分；
- 在书写属性时属性之间使用空格、换行等不影响属性的显示；
- 如果一个属性有几个值，则每个属性值之间用空格分隔。

2. 选择符

选择符中常用的是通配选择符、类型选择符、包含选择符、ID 选择符、类选择符和选择符分组。

1）通配选择符

通配选择符的写法是"＊"，其含义就是所有元素。例如：

```
*{font-size:12px}
```

表示页面中所有文本的字体大小均为 12 个像素。

2）类型选择符

类型选择符就是以文档语言对象类型作为选择符，即使用结构中的元素名称作为选择符。所有的页面元素都可以作为类型选择符，例如 body、div、p 等。

```
div{font-size: 12px; }
```

该样式实现的效果是页面中所有 div 元素包含的内容的字体大小为 12 个像素。

3）包含选择符

包含选择符的格式如下：

```
选择符 1 选择符 2
```

选择符 1 与选择符 2 之间用空格分隔，含义是所有选择符 1 中包含选择符 2。例如：

```
div p{font-size: 12px}
```

该样式实现的效果是，在所有被 div 元素包含的 p 元素中文本的字体大小均为 12 个像素。

4）ID 选择符

ID 选择符的格式如下：

```
#name
```

ID 选择符的语法格式是“#”加上自定义的 ID 名称。例如：

```
#name{font-size: 12px}
```

该样式实现的效果是，在所有调用 ID 名称为 name 的页面元素中文本的字体大小为 12 个像素。用户需要注意的是，ID 选择符的名称在页面中是唯一的。如果在页面中定义了 ID 选择符的名字为 name，则页面中其他 ID 选择符的名称不能以 name 命名。

5）类选择符

类选择符的格式如下：

```
.name
```

类选择符的语法格式是“.”加上自定义的类名称。例如：

```
.name{font-size: 12px}
```

该样式实现的效果是，在所有调用类名称为 name 的页面元素中文本的字体大小均为 12 个像素。类选择符的名称在页面中不是唯一的，可以通过定义相同的类名来调用同一个样式。

6）选择符分组

当多个选择符应用相同的样式时，可以将选择符用英文逗号分隔的方式合并为一组：

```
选择符 1,选择符 2,选择符 3
```

例如：

```
.name,div,p{font-size: 12px}
```

该样式实现的效果是，在类名字为 name 的元素、div 元素、p 元素中文本的字体大小为 12 个像素。

3. 属性

属性是 CSS 中最重要的部分，也是最复杂的部分，常用的属性有字体属性、文本属性、背景属性、定位属性、尺寸属性、布局属性、边界属性、边框属性、补白属性、列表项目属性、表格属性。其中，某些属性只有部分浏览器支持，这使属性的应用变得更加复杂。属性的应用是 CSS 应用的主体部分，请大家在使用过程中特别关注。

4. 伪类和伪元素

伪类和伪元素也是一种选择符，在页面元素中用来定义超出结构所能标识的样式。伪类是能被支持 CSS 的浏览器自动识别的特殊选择符。伪类的语法结构如下：

```
选择符伪类{属性: 属性值}
```

例如：

```
a: hover{font-size: 12px; }
```

该样式实现的效果是，当鼠标悬停在带有链接的文本上时，文本字体大小为 12 个像素。

伪类和伪元素一般是以"："开头。与类不同的是，伪类和伪元素在 CSS 中是指定的，不能随意命名和定义。

4.2.4 CSS 样式属性

CSS 样式属性是 CSS 应用的重点，下面简单介绍本系统中使用的一些基本布局属性。

1. 定位属性

定位属性包括 3 个方面，即定位模式(position)、边偏移(top、right、bottom、left)和层叠定位属性(z-index)。

1) 定位模式

它是一个不可继承的属性。其语法结构如下：

```
position:static | relative | absolute | fixed;
```

- static：元素按照普通方式生成，即元素按照 HTML 的规则进行定位；
- relative：元素将保持原来的大小偏移一定的距离；
- absolute：元素将从页面元素中被独立出来，使用边偏移进行定位；
- fixed：元素将从页面元素中被独立出来，但其位置是相对于屏幕本身，而不是文档的本身。

2) 边偏移

边偏移包括 top、right、bottom 和 left 4 个属性。

- top：定义元素相对于其父元素上边线的距离；
- right：定义元素相对于其父元素右边线的距离；
- bottom：定义元素相对于其父元素下边线的距离；
- left：定义元素相对于其父元素左边线的距离。

其语法结构如下(以 top 属性为例)：

```
top: auto |长度值|百分比值
```

3) 层叠定位属性

层叠定位属性(即 z-index 属性)用来定义元素的层叠顺序。其语法结构如下：

```
z - index:数字值
```

数字值为没有单位的数字值，并且可以取负数值。

2. 背景属性

页面背景的设置一般包括背景颜色、背景图片、背景颜色和背景图片混合等多种方式，这些方式也同样适用于页面中元素的背景。

1) 背景色

其语法结构如下：

```
background - color:颜色值;
```

2）背景图片

其语法结构如下：

```
background - image:url(图片路径);
```

3）背景图片的重复

背景图片的重复属性(即 background-repeat 属性)是不可以继承的属性。其语法结构如下：

```
background - repeat:repeat | no - repeat | repeat - x | repeat - y;
```

- repeat：背景图片按照从左到右、从上到下的顺序进行排列；
- no-repeat：背景图片不重复，在没有定义位置时，默认出现在容器的左上角；
- repeat-x：背景图片横向排列，在没有定义位置时，在容器顶部从左向右重复排列；
- repeat-y：背景图片纵向进行排列，在没有定义位置时，在容器左侧从上向下重复排列。

4）背景图片位置

背景图片的位置属性是不可继承的。其语法结构如下：

```
background - position:长度值|百分比值|top |right | bottom | left | center;
```

- 长度值和百分比值：背景图片按照设置的具体数值确定位置；
- top：背景图片出现在容器的上边；
- bottom：背景图片出现在容器的底边；
- left：背景图片出现在容器的左边；
- right：背景图片出现在容器的右边；
- center：背景图片横向和纵向居中。

3. 容器属性

在 CSS 中，所有的文档元素(包括块元素和内联元素)都会生成一个矩形框。这个矩形框由边界、边框、补白、宽度和高度几个部分组成。

1）补白属性

在 CSS 中，补白属性是一个不能继承的属性。其语法结构如下：

```
padding:长度值|百分比值;
```

2）边框属性

边框属性包括边框样式属性、边框宽度属性、边框颜色、单侧的边框等。

(1) 边框样式：边框样式属性是一个不可继承的属性。其语法结构如下：

```
border - style:none | hidden | dotted | dashed | solid |
            double | groove | ridge | inset | outset;
```

(2) 边框宽度：边框宽度属性是一个不可继承的属性。其语法结构如下：

```
border - width:medium | thin | thick |长度值;
```

(3) 边框颜色:边框颜色属性是一个不可继承的属性。其语法结构如下：

```
border - color:颜色值;
```

3）边界属性

边界属性是一个不可继承的属性。其语法结构如下：

```
margin:auto |长度值|百分比值;
```

4.3　主要页面设计

根据第 3 章中系统的总体规划，用户使用该办公自动化管理系统要经过系统首页→用户登录→系统主页 3 个页面转向过程。常用的办公自动化系统一般不设系统首页，而是将系统首页与登录界面合二为一。设置系统首页也有其优点，可以在这里对系统功能、使用方法以及注册登录等相关注意事项进行说明。下面完成“系统首页”“用户登录”以及“系统主页”页面的设计与显示。

4.3.1　系统首页设计

1. 添加文件夹

在第 2 章中，我们通过 Zend Studio 集成开发环境创建了名为 wmProject 的 Zend Framework 项目。现在，在该项目的 public 目录中添加文件夹 css、js、images，分别用来存放 CSS 样式文件、JavaScript 脚本文件以及图片文件。

在 Zend Studio 集成开发环境左侧的工作区中单击 public 文件夹，右键打开快捷菜单，选择 New|Folder 菜单项，进入新建文件夹对话框，输入文件夹名称，单击 Finish 按钮，完成文件夹的创建。

为了操作方便，我们在新创建的 css 文件夹下再创建一个名为 img 的文件夹，专门存放作为背景的图片资源。

2. 添加文件

在创建的 css 文件夹中添加 layout. css 和 style. css 样式文件。

在 Zend Studio 集成开发环境左侧的工作区中选择 public 目录下的 css 文件夹，右键打开快捷菜单，选择 New|CSS File 菜单项，打开新建文件对话框，输入文件名称，单击 Finish 按钮，完成样式文件的添加。然后双击打开样式文件，添加页面样式代码。

3. 添加图片资源

将准备好的图片资源放入 images 或 css 目录下的 img 文件夹中。在第 2 章中介绍过，Zend Studio 集成环境支持工作区的直接复制与粘贴，将图片复制到剪贴板上，然后直接粘贴到这两个文件夹中即可。

4. 添加页面视图代码

系统主页采用第 2 章中图 2.10 所示的系统欢迎页面，该页面的控制器与方法是由 ZF Tool 工具命令自动生成的。

打开系统主页视图文件 application\views\scripts\index\index. phtml，删除文件原有的全部内容，添加如下代码：

```
<?php echo $this->docType(); ?>
<html>
```

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<link href="/css/layout.css" rel="stylesheet" type="text/css" />
<link href="/css/style.css" rel="stylesheet" type="text/css" />
<?php echo $this->headTitle();?>
<style type="text/css">
  a{outline:none}
  img{border-style:none;}
</style>
</head>
<body id="mainBg">
<div id="wrap">
  <div id="mainImg01"></div>
  <div id="mainImg02">
      <a href="/user/login" >
         <img alt="单击进入" src="/images/enter.png" />
      </a>
   </div>
   <div id="mainImg03"></div>
</div>
</body>
</html>
```

上述代码非常简单，由最基本的网页元素组成，需要注意的是代码中的 3 处阴影部分。阴影代码的第 1、2 两个部分设置该页面的文档类型与标题，在本章的 4.1.1 节中设置了视图对象的"文档类型"与"文档标题"两个属性，这里通过 Zend Framework 的视图助手(占位符助手)在视图页面中输出这些设置。阴影代码的第 3 部分是从主页进入用户登录页面的超级链接，链接中的 href 属性值为/user/login，表示单击该链接，页面会跳转到 User 控制器的 login 方法视图页面。

打开 XAMPP 控制面板，启动 Apache 服务器，在浏览器的地址栏中输入 http://wmoams.com，系统首页效果如图 4.1 所示。

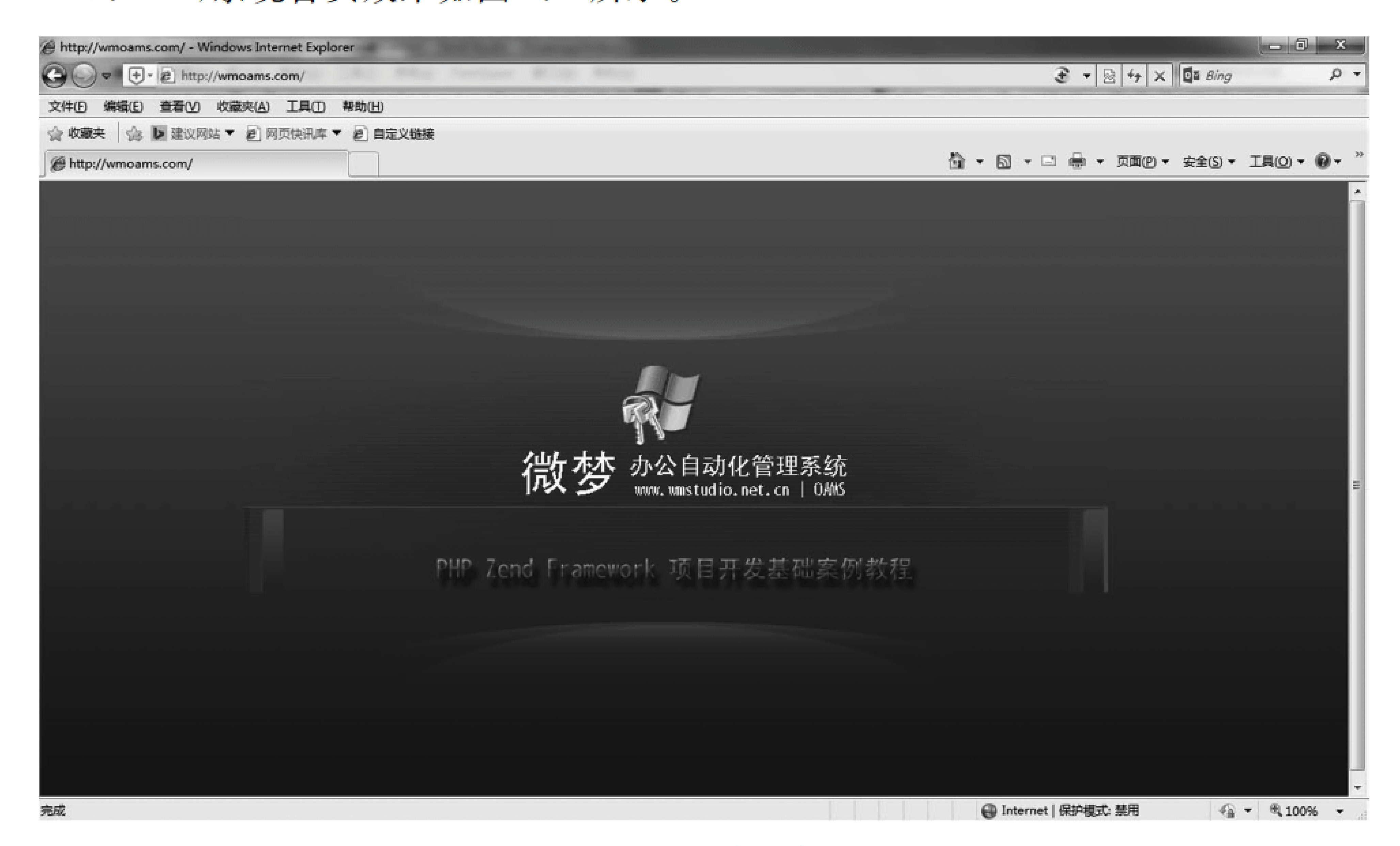


图 4.1　系统首页效果图

4.3.2 登录页面设计

1. 创建控制器

在第 2 章中学习了 Zend Framework 项目的结构与运行机制，了解了它的 MVC 设计模式与路由规则。Zend Framework 应用的所有视图页面都必须由相应的控制器进行管理，要在特定的页面中显示数据，必须创建相应的控制器与方法。

打开 Zend Studio 集成开发环境，单击 wmProject 项目中的任何一个文件夹或文件。然后选择 Project|Zend Tool 菜单项，打开 ZF Tool 命令工具窗口，输入命令：

```
zf create controller User
```

在默认模块 default 中创建一个名为 User 的控制器。其中，User 为自定义的控制器名字；控制器名字的首字母可以大写也可以小写，若为小写，ZF Tool 工具会自动将其改为大写后生成名为 UserController 的控制器类。控制器名字最好用单个单词，若非要用多个单词，最好用下划线"_"连接，以免 Zend Framework 框架在搜索资源时出现错误。

根据 ZF Tool 命令行语法格式，User 控制器名的后面应该是控制器所属的模块名，这里省略了该参数，即为默认模块，默认模块名为 default。

上述命令生成的 User 控制器文件 UserController. php 位于 application\controllers 目录下，其代码如下：

```
class UserController extends Zend_Controller_Action
{
    public function init()
    {
        /* Initialize action controller here */
    }
    public function indexAction()
    {
        //action body
    }
}
```

从上述代码可以看出，所谓的"控制器"实际上是一个继承于 Zend_Controller_Action 的类，并且默认生成了 init 初始化方法与 index 方法。init 方法用于 User 控制器中所有方法(action)的初始化，它不对应视图(View)，而 index 方法对应相应的视图。

从项目目录中可以看到，在 application/views/scripts 目录中，ZF Tool 工具自动生成了一个名为 User 的文件夹，这个文件夹中存放的就是 User 控制器中的所有方法对应的视图文件。现在 User 控制器中还只有名为 index 的方法，所以文件夹中只有一个名为 index. phtml 的页面文件。注意，视图文件夹名与控制器名相同，视图文件名与方法名相同，视图文件的扩展名为 phtml。

2. 添加方法

为了能通过方法的名称推测其功能，下面创建一个名为 login 的方法来处理用户登录请求。

在 Zend Studio 集成开发环境中使用与上面相同的方法打开 ZF Tool 命令窗口，输入命令：

```
Zf create action login User
```

在 User 控制器中创建一个名为 login 的方法。上述命令中的 login 为方法名，User 为控制器名。打开 application\controllers\UserController.php 文件，可以看到 ZF Tool 工具创建的 loginAction 方法。它是 UserController 控制器类的一个成员函数。

上述命令还在 application\views\scripts\user 目录下创建了与 login 方法同名的视图文件 login.phtml。

3. 设计登录视图

系统用户登录页面结构如图 4.2 所示。



图 4.2 用户登录页面效果图

从图 4.2 可以看出，在垂直方向上用户登录页面可以分为 3 个区域：顶部为企业 logo 及系统标题，底部为页面脚标；中部为用户登录区，登录区的右边是登录表单设置区。

由于本系统是基于 Zend Framework 框架的，我们将采用 Zend_Form 表单实现登录操作。Zend_Form 表单具有多项验证功能，能够增强系统的安全性，详细内容将在第 6 章中介绍。

打开 application\views\scripts\user\login.phtml 视图文件，删除原有全部内容，添加如下代码：

```
<?php echo $this->docType(); ?>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<link href="/css/layout.css" rel="stylesheet" type="text/css" />
<link href="/css/style.css" rel="stylesheet" type="text/css" />
<?php echo $this->headTitle();?>
</head>
<body id="mainBg">
```

```
<div id="header" style="border-bottom:1px solid #cccccc ">
    <div id="logo"><img src="/images/logo.jpg"></div>
</div>
<div id="wrap">
    <div id="login_center">
        <div class="left">
           <img src="/images/login_center_mark.gif" /></div>
        <div class="center"></div>
        <div class="right"></div>
    </div>
</div>
<div id="header" style="border-top:1px solid #cccccc">
<span id="footerText">微梦软件 - 微梦办公自动化管理系统
                                 http://www.wmstudio.net.cn</span></div>
</body>
</html>
```

4.3.3 系统主页设计

1. 系统主页结构

用户登录成功后进入系统主页,如图 4.3 所示。



图 4.3 系统主页效果图

根据用户对各个功能模块的使用频率和重要程度,本系统的首页页面中的显示模块主要分为以下 5 个部分。

- 系统首页导航区:包括系统导航菜单、当前用户、时间等;
- 系统左侧导航区:包括用户常用菜单;
- 系统主显示区:默认显示通知公告、最新公文、个人日程以及最新消息;
- 系统右侧提示区:默认显示待办事务与近期工作安排;

• 系统底部显示区：显示版权等信息。

2. 添加方法

在上面的演示中，系统首页页面对应的是 Index 控制器的 index 方法，所以需要在 Index 控制器中为系统主页另外添加一个 main 方法。

在 Zend Studio 集成开发环境中使用与上面相同的方法打开 ZF Tool 命令窗口，输入命令：

```
Zf create action main Index
```

在 Index 控制器中创建名为 main 的方法。

3. 主页页面设计

对于系统的内部页面，不同的数据类型应具有不同的布局方式，但有些公共的或特别重要的内容需要时时刻刻显示在页面上。例如，在如图 4.3 所示的系统主页中，左、右两侧框架中所显示的内容就需要在系统的每个页面上都能显示出来。如果不采取某些技术措施，需要在系统的每个页面上添加这些相同的代码，这显然是不合适的。为此，我们启用 Zend Framework 框架的布局模板，所有信息都在这个模板页面上显示出来，这样既有利于代码的更新，也保持了整个系统页面的统一性。

先看一下系统内部页面的布局模板效果，如图 4.4 所示。

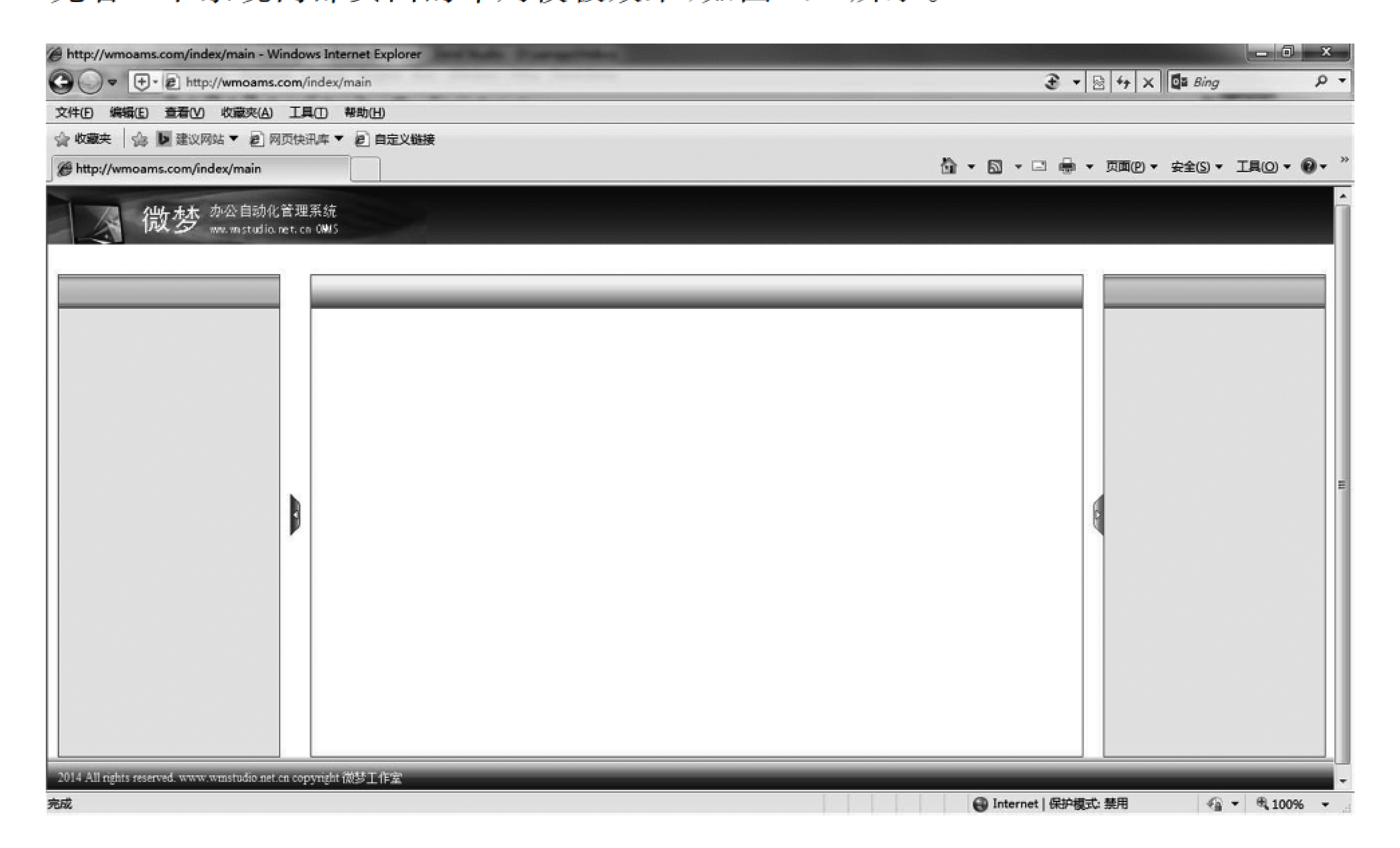


图 4.4 布局模板页面效果图

1）开启模板功能

在 Zend Studio 集成开发环境的工作区中选择 wmProject 项目，定位到项目内部的文件或文件夹上，选择 Project|Zend Tool 菜单项，打开 Zend Tool 工具命令窗口，输入命令：

```
Zf enable layout
```

开启 Zend Framework 的布局模块功能。若命令执行成功，打开 application 文件夹，就能够

看到上述 Zend Tool 命令创建的 layout 文件夹和 layout.phtml 页面文件。另外,在 application\configs\application.ini 中,也可以看到多了一条如下的配置代码,它定义了 layout 模板布局文件所在的路径,即 layout.phtml 文件路径。

```
resources.layout.layoutPath = APPLICATION_PATH "/layouts/scripts/"
```

布局模板开启后,系统中所有的视图内容都将通过 layout.phtml 页面显示出来,当然也包括系统界面首页以及用户登录页面。很显然,这两个页面用图 4.4 所示的结构显示是不合适的。通过下面的方法,可以在不必要的时候关闭 layout 布局模板。

```
$ this->_helper->layout->disableLayout();
```

将该代码分别添加于 Index 控制器的 Index 方法、User 控制器的 login 方法中,关闭这两个控制器方法所对应的视图页面、在显示时布局模板的使用,即直接显示这两个页面的视图文件内容,就像我们前面做的一样。

```
class IndexController extends Zend_Controller_Action
{
…
    public function indexAction()
    {
        $ this->_helper->layout->disableLayout();
    }
}
class UserController extends Zend_Controller_Action
{
…
    public function loginAction()
    {
        $ this->_helper->layout->disableLayout();
    }
}
```

代码中使用的_helper 是 Zend Framework 的"动作助手",详细内容将在第 11 章进行介绍。

2) 设计模板页面

分析如图 4.4 所示的模板页面结构,可以将其分成 5 个区,如图 4.5 所示。

3) 编写布局页面代码

在图 4.5 所示的页面结构中,除主显示区以外,其余 4 个显示区都是固定的。也就是说,这 4 个区中的内容是必须显示在每个页面上的。由此,我们把这 4 个显示区分别用 header.phtml、left.phtml、right.phtml 和 footer.phtml 4 个视图文件表示。

在 Zend Studio 项目工作区中选择 application\layouts\scripts 文件夹,右键打开快捷菜单,选择 New|PHP File 菜单项,打开创建新 PHP 文件对话框,输入文件名后单击 Finish 按钮完成。为了保持视图文件类型统一,注意将文件扩展名改为 phtml。

打开 application\layouts\scripts 文件夹中的 5 个视图文件,分别添加代码。



图 4.5　布局模板结构

（1）视图文件 layout.html 代码：

```
<?php echo $this->docType();?>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<link href="/css/layout.css" rel="stylesheet" type="text/css" />
<link href="/css/style.css" rel="stylesheet" type="text/css" />
<?php echo $this->headTitle();?>
</head>
<body>
<div id="wrap">
<?php
    echo $this->partial('header.phtml');
?>
<div id="main">
    <div id="sidebar"><?php echo $this->partial('left.phtml');?></div>
    <div id="leftframe"></div>
    <div id="content"></div>
    <div id="right"><?php echo $this->partial('right.phtml');?></div>
    <div id="rightframe"></div>
</div>
    <?php echo $this->partial('footer.phtml'); ?>
</div>
</body>
</html>
```

（2）视图文件 header.phtml 代码：

```
<div id="header">
    <div id="logo"><img src="/images/logo.jpg"></div>
</div>
```

(3) 视图文件 footer.phtml 代码：

```
<div id="footer">
    2014 All rights reserved. www.wmstudio.net.cn copyright 微梦工作室
</div>
```

(4) 视图文件 left.phtml 和 right.phtml 代码见源代码。

4.4 layout 布局模板

上面介绍了 layout 布局模板的开启、关闭方法及其视图文件结构，下面较为详细地说明它的特点及一些常用方法的使用。

4.4.1 布局模板概述

Zend Framework 的 Zend_Layout 组件实现 layout 布局模板功能。Zend_Layout 类实现了经典的两步视图(two step view)模型，允许把应用程序的内容包装在另一个视图中，所以一般作为系统的模板使用。

如果要更多地了解 Zend Framework 的布局模板，读者可以查看 Zend Framework 使用手册或直接查看 Zend Framework 库文件中的 zend\layou.php 源文件。

在 Zend Studio 集成开发环境中打开库文件 layout.php，查看类 Zend_Layout 的定义，其成员变量及成员函数如图 4.6 所示。

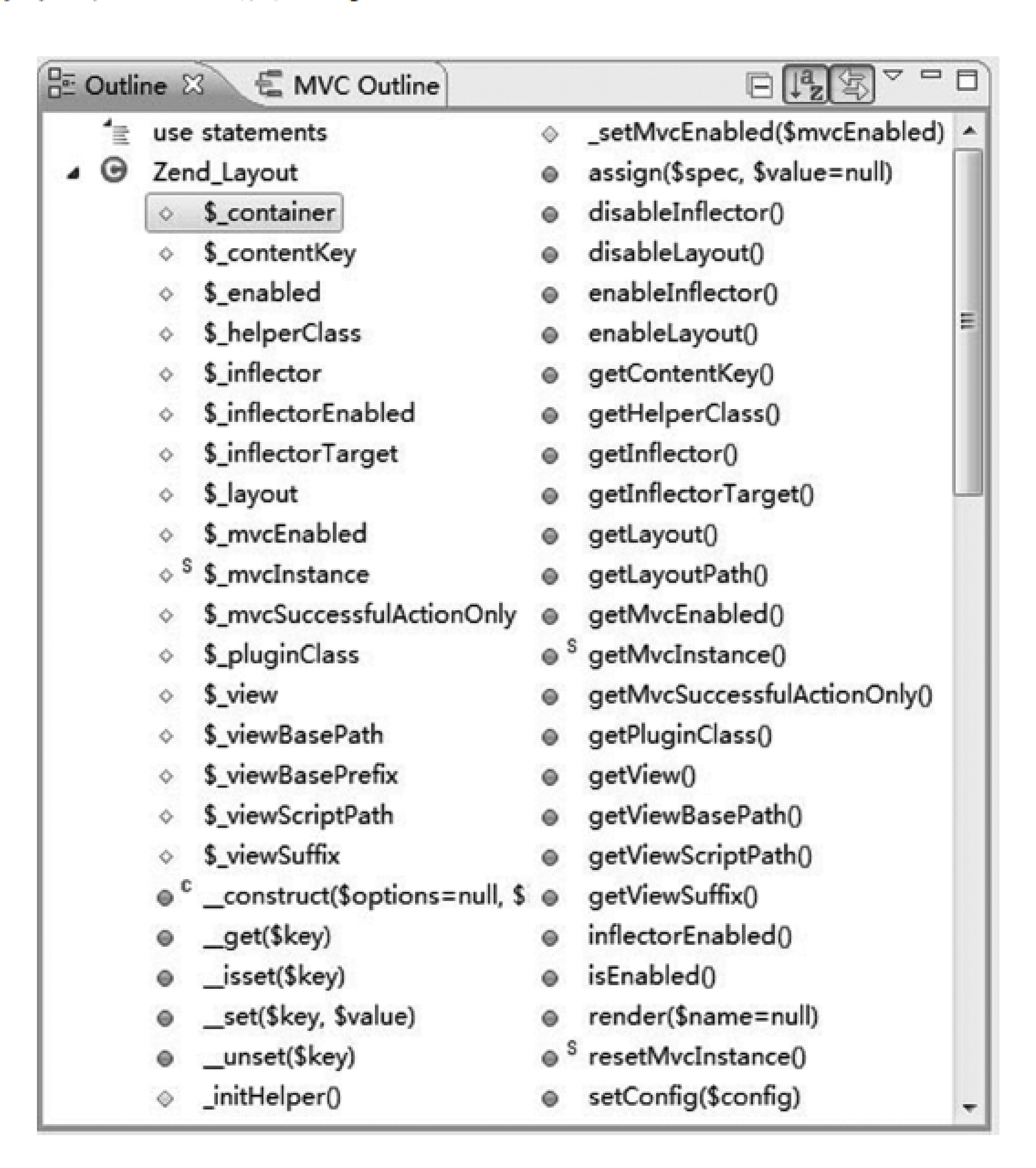


图 4.6 类 Zend_Layout 成员变量及方法

在 Zend Framework 中开启 layout 布局，实际上就是调用了 Zend_Layout 类的 startMvc() 方法，Zend_Layout 自动在 front controller 中注册一个 Zend_Layout_Controller_Plugin_Layout 的插件资源，就像前面在配置文件中所看到的 resources.layout 那样。这个插件类

中的 postDispatch 方法负责把视图文件(script)的内容放到模板(layout)的预留位置中,一般默认名为 content,此即为两步视图在 Zend Framework 中实现的关键。

4.4.2 布局模板的关闭

图 4.6 中列出了 zend_Layout 类的一些常用方法,通过这些方法可以对 layout 模板布局进行操作。在前面的操作中使用代码:

```
$ this -> _helper -> layout -> disableLayout();
```

来关闭布局模板,其中的方法 disableLayout 实现布局模板的关闭。

从这里也可以看出,代码中的 layout 应该是 Zend_Layout 类的实例(对象)。其中的 _helper 是 Zend Framework 框架的动作助手,通过它可以很方便地完成一些操作。

4.4.3 多个布局模板的使用

从图 4.6 列出的类 Zend_Layout 方法中可以找到一个名为 setLayout 的方法,这个方法是用来设置新布局模板的。其源代码如下:

```
public function setLayout( $ name, $ enabled = true)
    {
        $ this -> _layout = (string) $ name;
        if ( $ enabled) {
            $ this -> enableLayout();
        }
        return $ this;
    }
```

调用该方法需要带一个或两个参数,name 为新布局的名字,即新布局文件 *.phtml 的文件名。该方法中调用了 enableLayout 方法启用布局模板。

【例 4.1】 在项目 wmProject 中使用新模板 admin。

(1) 设计新布局模板文件。

在 Zend Studio 集成开发环境中选择工作区中的 wmProject 项目下的 application\layouts\scripts 文件夹,右击该文件夹新建文件 admin.phtml,并添加代码:

```
<?php echo $ this -> docType();?>
< html >
< head >
< meta http - equiv = "Content - Type" content = "text/html; charset = utf - 8" />
< link href = "/css/layout.css" rel = "stylesheet" type = "text/css" />
< link href = "/css/style.css" rel = "stylesheet" type = "text/css" />
<?php echo $ this -> headTitle();?>
</head >
< body >
< div id = "wrap">
<?php   echo $ this -> partial('header.phtml'); ?>
< div id = "main">
    < div id = "sidebar"></div >
    < div id = "content" style = "width:78 % ">
```

```
<?php echo $this->layout()->content; ?>
</div>
</div>
    <?php echo $this->partial('footer.phtml'); ?>
</div>
</body>
</html>
```

在上述阴影部分代码中，layout 是 Zend Framework 的视图助手，content 是启用了该布局模块的控制器所对应的视图文件内容。如果在 User 控制器的 index 方法中启用 admin 布局模板，则 content 即代表 application\views\scripts\user\index.phtml 视图页面内容。

(2) 在控制器的方法中添加代码，启用新布局模板。

打开 User 控制器的 index 方法，添加如下代码：

```
class UserController extends Zend_Controller_Action
{
…
    public function indexAction()
    {
        $this->_helper->layout->setLayout('admin');
    }
…
}
```

(3) 在浏览器的地址栏中输入 http://wmoams.com/User，页面效果如图 4.7 所示。

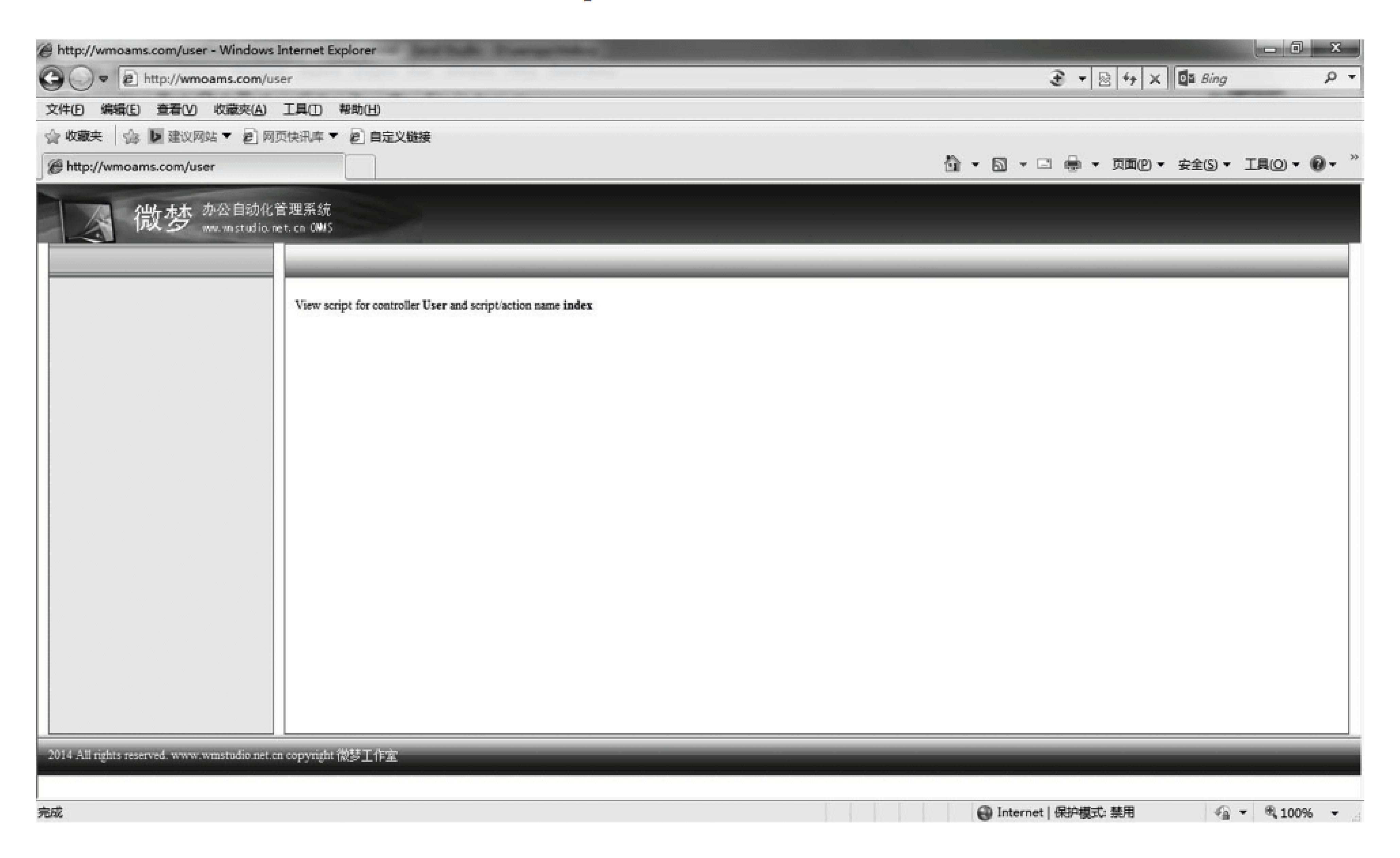


图 4.7 新布局模板的页面效果

从图 4.7 中可以看出，新布局模板采用经典的左右页面形式，右主显示区中的文本来自 application\views\scripts\user\index.phtml 页面。这说明，要显示的视图内容被布局管理器控制在了布局中的一块区域，这个区域的内容可以动态显示。

4.4.4 布局文件目录的更改

在开启 Zend Framework 的 layout 布局模板后会生成默认的布局文件目录，即 application\layouts\scripts 目录。这个目录在 application\configs\application.ini 文件中被定义，可以通过 Zend_Layout 类的 setLayoutPath 重新设置布局文件路径，这样可以对布局文件进行分类管理。

【例 4.2】 将上述新布局文件 admin.phtml 存放在目录 mylayouts 中。

(1) 在项目 wmProject 的 application 下新建文件夹 mylayouts\scripts。

(2) 将布局文件 admin.phtml 复制到新文件夹中，删除原 layouts\scripts 文件夹中的布局文件。

(3) 将 admin.phtml 文件中需要包含的文件(例如 header.phtml 和 footer.phtml)一并复制到新文件夹下。

(4) 在控制器的方法中添加代码，设置新布局文件的路径，假如还是用 User 控制器的 index 方法进行测试，则代码如下：

```
class UserController extends Zend_Controller_Action
{
    …
    public function indexAction()
    {
        $ this->_helper->layout->setLayout('admin');
        $ this->_helper->layout->setLayoutPath(
                                APPLICATION_PATH.'/mylayout/scripts');
    }
…
}
```

(5) 在浏览器地址栏中输入 http://wmoams.com/user，页面效果与图 4.7 相同，说明 Zend Fromwork 在新的路径下找到了新的布局文件。

4.4.5 布局文件名称的修改

在项目中开启布局模板后，Zend Framework 生成了一个默认的名为 layout.phtml 的布局文件。虽然可以用上面介绍的方法关闭、启用新布局，但当项目中的控制器比较多的时候，这种方法显得比较麻烦，并且修改起来也不方便。这时，可以使用 Zend Framework 提供的插件功能通过编写插件来改变 Zend Framework 框架的路由规则，从而更改默认布局页面文件的名称，关于这方面的知识，在后面的章节中再进行介绍。

4.5 本章小结

本章通过实现系统首页、用户登录以及系统主页页面进一步学习了 Zend Framework 项目的运行机制，使读者更直接地理解了 MVC 模式中控制器与视图之间的相互关系，同时以此为基础详细介绍了 Zend Framework 的 Zend_Layout 组件功能及使用方法。

Zend_Layout 组件是本章的重点，该组件实现了 Zend Framework 应用的 layout 两步视图模式，使得项目布局更加合理、模板代码更加简洁，大大提高了代码的可重用性，减少了冗余。

第5章 页面导航及 Zend_Db 数据库

页面导航和数据库操作是每个 Web 应用都不可缺少的，Zend Framework 框架提供了专门的组件来实现这两项功能。Zend Framework 的 Zend_Navigation 组件用于创建导航菜单、面包屑导航或网站地图，Zend_Db 组件使用类的封装机制实现了对数据库的各种操作。

本章通过案例项目的菜单制作以及新闻文章显示的实现，介绍 Zend Framework 中的页面导航及数据库操作。

5.1 导航菜单

Zend Framework 框架使用 Zend_Navigation 组件创建导航菜单，在使用这个组件时，需要 XML 或 INI 文件的配合。与传统的 HTML、CSS 或 JavaScrip 菜单相比，Zend_Navigation 的导航实现并不是很直观，但它拥有代码量少、动态功能强等诸多特性，所以使用 Zend_Navigation 创建导航菜单仍然是一个很经济的技术方案。

使用 Zend_Navigation 创建导航菜单一般分为 3 个步骤，首先创建一个 XML 文件，设置应用程序的菜单结构；然后在应用程序的启动文件 Bootstrap.php 中初始化 Zend_Navigation 组件，使其能够读取到相应的资源；最后在视图页面的适当位置显示导航菜单。

5.1.1 创建 XML 文件

可扩展标记语言 XML 是一种灵活地传递数据的标记语言，主要用来定义结构、存储信息、传送信息，而不是像 HTML 或 CSS 那样用来表现或展示数据。

启动 Zend Studio 集成开发环境，打开项目 wmProject，选择 application\configs 目录，右击该文件夹，在弹出的快捷菜单中选择 New|XML File 菜单项，在对话框中输入文件名 navigation.xml 及相关选项内容，然后单击 Finish 按钮完成新文件的创建。

双击打开 navigation.xml 文件，添加如下代码：

```
<?xml version = "1.0" encoding = "UTF - 8"?>
<configdata>
    <company>
        <label>公司主页</label>
        <uri>http://www.wmstudio.net.cn</uri>
    </company>
    <home>
        <label>首页</label>
```

```
        <controller> index </controller>
        <action> main </action>
    </home>
    <document>
        <label>公文管理</label>
        <controller> document </controller>
        <action> index </action>
        <pages>
            <issue>
                <label>发文管理</label>
                <controller> document </controller>
                <action> issue </action>
            </issue>
            <receive>
                <label>收文管理</label>
                <controller> document </controller>
                <action> receive </action>
            </receive>
            <interfile>
                <label>公司报告</label>
                <controller> document </controller>
                <action> file </action>
            </interfile>
            <readdoc>
                <label>文件传阅</label>
                <controller> document </controller>
                <action> readdoc </action>
            </readdoc>
        </pages>
    </document>
    <meeting>
        <label>会议管理</label>
        <controller> meeting </controller>
        <action> index </action>
        <pages>
            <issue>
                <label>会议日程</label>
                <controller> meeting </controller>
                <action> plan </action>
            </issue>
            <notice>
                <label>会议通知</label>
                <controller> meeting </controller>
                <action> notice </action>
            </notice>
        </pages>
    </meeting>
    <office>
        <label>事务管理</label>
        <controller> office </controller>
        <action> index </action>
```

```
        <pages>
            <leadership>
                <label>领导日程</label>
                <controller>office</controller>
                <action>leadership</action>
            </leadership>
            <request>
                <label>公用申请</label>
                <controller>office</controller>
                <action>request</action>
            </request>
            <vehicle>
                <label>车辆管理</label>
                <controller>office</controller>
                <action>vehicle</action>
            </vehicle>
            <stamper>
                <label>印章管理</label>
                <controller>office</controller>
                <action>stamper</action>
            </stamper>
            <holiday>
                <label>请假管理</label>
                <controller>office</controller>
                <action>holiday</action>
            </holiday>
            <repair>
                <label>报修管理</label>
                <controller>office</controller>
                <action>repair</action>
            </repair>
        </pages>
    </office>
    <personal>
        <label>个人办公</label>
        <controller>personal</controller>
        <action>index</action>
        <pages>
          <mymessage>
                <label>个人消息</label>
                <controller>personal</controller>
                <action>mymessage</action>
          </mymessage>
          <todoitem>
                <label>待办事宜</label>
                <controller>personal</controller>
                <action>todoitem</action>
          </todoitem>
          <doneitem>
                <label>已办事宜</label>
                <controller>personal</controller>
```

```xml
                <action>doneitem</action>
            </doneitem>
            <worklog>
                <label>工作日志</label>
                <controller>personal</controller>
                <action>worklog</action>
            </worklog>
            <myschedule>
                <label>个人日程</label>
            <controller>personal</controller>
            <action>myschedule</action>
              </myschedule>
              <email>
                <label>电子邮件</label>
            <controller>personal</controller>
            <action>email</action>
              </email>
              <addresslist>
                <label>通讯录</label>
                <controller>personal</controller>
                <action>address</action>
              </addresslist>
              <instruction>
                <label>常用批示</label>
                <controller>personalapp</controller>
                <action>instruct</action>
              </instruction>
              <personalinfo>
                <label>个人名片</label>
                <controller>personal</controller>
                <action>personalinfo</action>
              </personalinfo>
          </pages>
      </personal>
      <publicservice>
          <label>公共服务</label>
          <controller>pubservice</controller>
          <action>index</action>
          <pages>
              <publicview>
                <label>公共浏览</label>
                <controller>pubservice</controller>
                <action>pubview</action>
              </publicview>
              <govopen>
                <label>政务公开</label>
                <controller>pubservice</controller>
                <action>govopen</action>
              </govopen>
              <storage>
                <label>网络存储</label>
```

```
                <controller>pubservice</controller>
                <action>storage</action>
            </storage>
            <download>
                <label>资料下载</label>
                <controller>pubservice</controller>
                <action>download</action>
            </download>
            <express>
                <label>快递查询</label>
                <controller>pubservice</controller>
                <action>express</action>
            </express>
        </pages>
    </publicservice>
</configdata>
```

以上 XML 文件定义了 company、home、document、meeting、office、personal、publicservice 共 7 个标签，它们将成为一级导航菜单，然后在 document 下创建了一个名为 pages 的标签，其中又包含了 issue、receive、interfile 4 个标签，它们将成为 document 下的二级导航菜单。在每一个标签中都定义了 lable、controller 及 action，它们将分别成为菜单名称和菜单所指向的链接页面。

5.1.2 初始化 Zend_Navigation 组件

在 Zend Studio 集成开发环境中打开 application\Bootstrap.php 启动文件，在 Bootstrap 类中添加_initNavigation()初始化函数，并编写如下代码：

```
class Bootstrap extends Zend_Application_Bootstrap_Bootstrap
{
…
protected function _initNavigation()
    {
        $this->bootstrap('layout');
        $layout = $this->getResource('layout');
        $view = $layout->getView();
        $config = new Zend_Config_Xml(
                          APPLICATION_PATH.'/configs/navigation.xml');
        $navigation = new Zend_Navigation($config);
        $view->navigation($navigation);
    }
}
```

上述代码分 4 个步骤创建页面导航。首先分别调用 Bootstrap 类的 bootstrap()方法和 getResource()方法，启用 layout 布局模板并获取模板对象；然后调用 Zend_Layout 类的 getView()方法，得到视图对象；接下来实例化一个 Zend_Config_Xml 对象，在实例化时使用前面创建的 navigation.xml 文件的路径字符串作为参数，读取 navigation.xml 中的配置，紧接着实例化一个 Zend_Navigation 对象，并将对象 config 作为参数读入其中；最后将对象

Navigation 添加到 View 视图中，这样应用程序在渲染 layout 视图模板页面时就会调用 navigation. xml 文件中定义的标签生成导航菜单。

5.1.3 显示导航菜单

1. 输出导航菜单

在 layout 布局模板视图的主显示区顶部渲染输出导航菜单。在模板文件 application\layouts\layout. phtml 中添加代码：

```
…
<body>
<div id="wrap">
<?php echo $this->partial('header.phtml'); ?>
<div id="nav"><?php echo $this->navigation()->menu(); ?></div>
…
</body>
```

这一行代码很简单，直接用 echo 输出在 Bootstrap 启动文件中配置的 navigation 资源，菜单效果如图 5.1 所示。

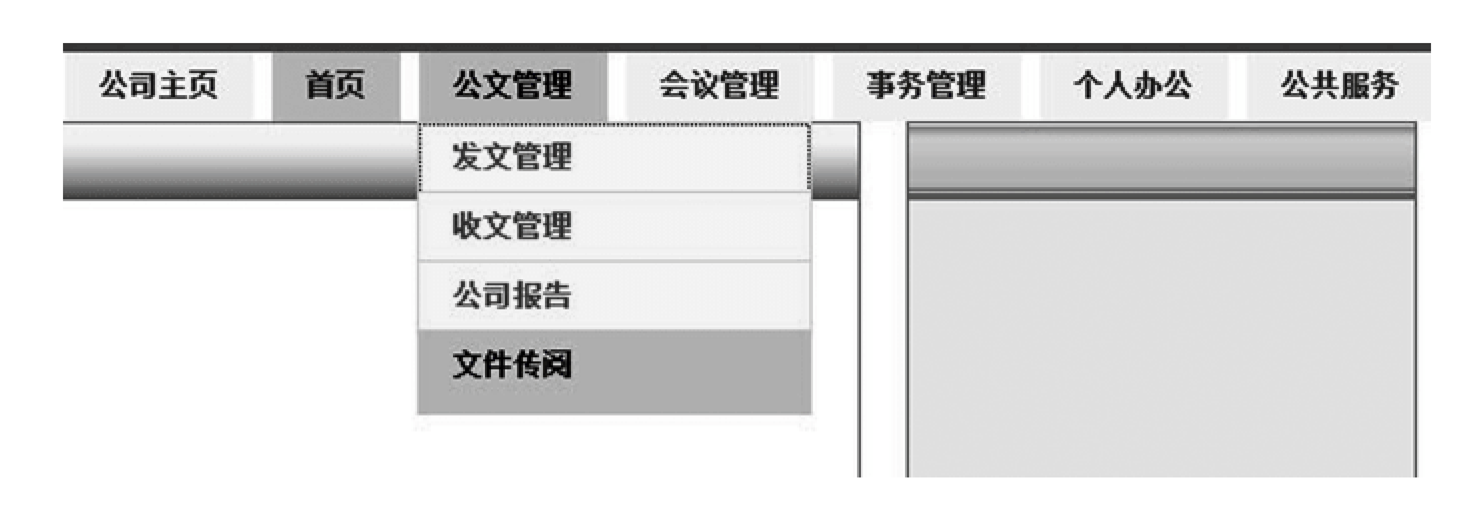


图 5.1 资源

2. 输出面包屑导航

面包屑导航(Breadcrumb Navigation)这个概念来自童话故事“汉赛尔和格莱特”，当汉赛尔和格莱特穿过森林时，不小心迷路了，但是他们发现在沿途走过的地方都撒下了面包屑，这些面包屑帮助他们找到了回家的路。所以，面包屑导航就是告诉用户目前在系统中的位置以及如何返回的一种技术方案。

打开模板文件 application\layouts\layout. phtml，在页面中添加代码：

```
…
<body>
<div id="wrap">
<?php echo $this->partial('header.phtml'); ?>
<div id="breadnav">
    <span>您目前的位置：</span><?php echo $this->navigation()->
breadcrumbs()->setLinkLast(false)->setMinDepth(0)->render(); ?>
</div>
<div id="nav"><?php echo $this->navigation()->menu(); ?></div>
…
</body>
```

其中的 setLinkLast()方法定义导航的最后一项是否有链接，其参数是 false 或 true；

setMinDepth 是导航出现的页面层级，设置为 1，表示只在一级分类页面以下的页面显示；如果设置为 0，则在首页中也会显示面包屑，如图 5.2 所示。

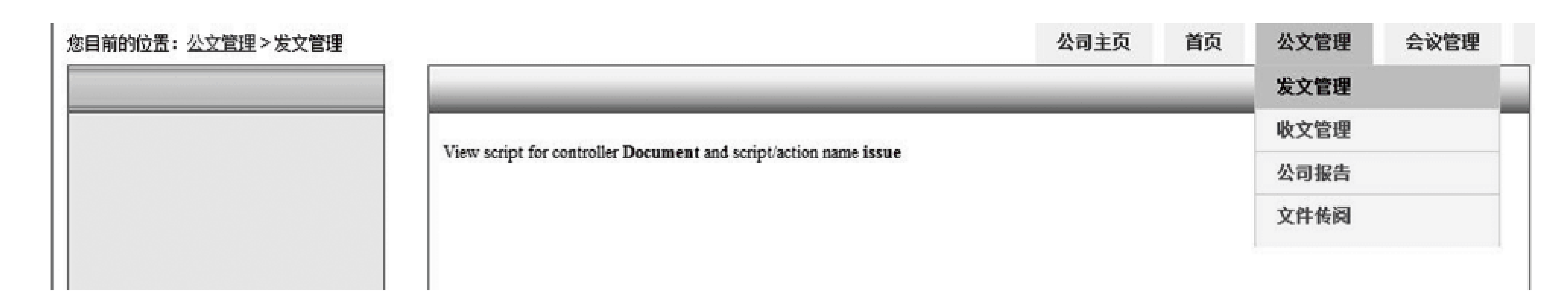


图 5.2　面包屑导航效果

该图中上部显示的“公文管理”|“发文管理”即为面包屑导航，它是用户选择“公文管理”下的“发文管理”菜单项后显示的效果，说明用户现在位于系统的“发文管理”信息显示页面位置。当系统页面层级较多时，能够让用户非常清楚地知道自己目前的位置，并能快速地返回到上一级页面。

5.2　Zend_Navigation 组件

上面演示了 Zend Framework 的 Zend_Navigation 组件的功能及使用方法，下面简单分析一下原理，以使大家在使用时能够更加灵活。

从上面的实例可以看出，导航菜单的创建是在 Bootstrap 启动文件中通过加载 navigation.xml 文件完成的。所谓的“导航菜单”，实际上就是一个 Zend_Navigation 类的实例。

打开 Zend Framework 库文件中 Zend_Navigation 类的定义文件，其源代码如下：

```
class Zend_Navigation extends Zend_Navigation_Container
{
    public function __construct( $pages = null)
    {
        if (is_array( $pages) || $pages instanceof Zend_Config) {
            $this->addPages( $pages);
        } elseif (null !== $pages) {
            require_once 'Zend/Navigation/Exception.php';
            throw new Zend_Navigation_Exception(
                    'Invalid argument: $pages must be an array, an ' .
                    'instance of Zend_Config, or null');
        }
    }
}
```

可以看出，Zend_Navigation 类继承于类 Zend_Navigation_Container，实例化时需要带一个参数 pages。从接下来的 if 语句可知，pages 参数既可以是 Zend_Config 类实例，还可以是数组。上面创建的 navigation.xml 便属于 Zend_Config 实例，当然，这个 XML 文件也可以用 INI 配置文件的形式，就像应用程序的 application.ini 一样。

5.2.1　Zend_Navigation_Page 类

实际上，创建 Zend_Navigation 实例时带进的参数 pages 是 Zend_Navigation 的一个 Zend_Navigation_Page 对象。Zend_Navigation_Page 类有两个子类，即 Zend_Navigation_

Page_Mvc 和 Zend_Navigation_Page_Uri，它们分别对应 MVC page 和 URI page 两种页面类型。

1. 页面的通用属性

Zend_Navigation 导航菜单中所指向的页面不管是哪种类型都具有如下公共属性。

- label：链接标签；
- id：链接 ID；
- class：类样式；
- title：窗口标题；
- target：窗口类型。

上面只列出了部分属性，更多属性请参考 Zend Framework 使用手册。这些属性实际上就是 HTML 中<a>标签中的属性，对于它们的含义大家都比较熟悉，这里不再赘述。

2. MVC pages

MVC pages 是定义于 Zend_Controller MVC 模式下的一种页面类型，通过 Zend_Navigation_Page_Mvc 类创建，它除了具有上面的公共属性之外，还具有符合 MVC 路由特点的一些属性。

- Action：页面所属的方法名；
- Controller：页面所属的控制器名；
- Module：页面所属的模块名；
- Params：URI 中的附加参数。

分析一下前面用 XML 文件定义的每个菜单项，例如"首页"菜单项：

```
<home>
        <label>首页</label>
        <controller>index</controller>
        <action>main</action>
 </home>
```

可以发现，设置了 action 和 controller 属性值。很明显，这些菜单所指向的页面都属于 MVC pages 页面，它的转向符合 MVC 模式规则。

用户还可以用如下形式创建 MVC pages：

```
//菜单"首页"指向 /index/main 页面
$mainpage = new Zend_Navigation_Page_Mvc(array(
    'label'   => '首页',
    'action'     => 'index',
    'controller' => 'main'
));

//菜单"ZendBog"指向 /blog/post/view/id/1337 页面
$page = new Zend_Navigation_Page_Mvc(array(
    'label'      => 'ZendBog',
    'action'     => 'view',
    'controller' => 'post',
    'module'     => 'blog',
    'params'     => array('id' => 1337)
));
```

3. URI pages

上面的 MVC pages 页面指向的是应用内部，而 URI pages 能让菜单指向应用外的某个链接，除了上面的公共属性外，它只有 uri 一个特有属性，用来指明链接地址。

```
$page = Zend_Navigation_Page_Uri (array(
    'label' => '百度',
    'uri'   => 'http://www.baidu.com/'
));
```

4. 效果测试

将上述 2、3 中的代码添加到引导文件 Bootstrap.php 中进行测试，效果如图 5.3 所示。



图 5.3　菜单导航示例效果

```
protected function _initNavigation()
    {
        $this->bootstrap('layout');
        $layout = $this->getResource('layout');
        $view = $layout->getView();

        //菜单"首页"指向 /index/maind 页面
        $mainpage = new Zend_Navigation_Page_Mvc(array(
                'label'      => '首页',
                'action'     => 'index',
                'controller' => 'main'
        ));
        //菜单"ZendBog"指向 /blog/post/view/id/1337 页面
        $page1 = new Zend_Navigation_Page_Mvc(array(
                'label'      => 'ZendBog',
                'action'     => 'view',
                'controller' => 'post',
                'module'     => 'blog',
                'params'     => array('id' => 1337)
        ));
        $page2 = new Zend_Navigation_Page_Uri (array(
                'label' => '百度',
                'uri'   => 'http://www.baidu.com/'
        ));
        $navigation = new Zend_Navigation();
        $navigation->addPages(array($mainpage, $page1, $page2));
        $view->navigation($navigation);
    }
}
```

观察菜单效果时，请注意将鼠标指针移动到某菜单项时浏览器状态栏上显示的链接地址。

5.2.2　Zend_Navigation_Container 类

从 Zend_Navigation 类的定义可知，Zend_Navigation_Container 类是其父类。也就是说，我们创建的导航菜单同样是一个 Zend_Navigation_Container 类的实例。除了可以使用

配置文件创建 Zend_Navigation_Container 实例外，还可以使用数组创建 Zend_Navigation_Container 实例，例如：

```
$ container = new Zend_Navigation(array(
    array(
          'label' => '百度',
          'uri' => 'http://www.baidu.com/'
          ),
     array(
          'label' => '新闻动态',
          'controller' => 'news',
          'pages' => array(
               array(
                    'label' => '学校新闻',
                    'action' => 'unvnews',
                    'controller' => 'news'
                    ),
               array(
                    'label' => '学院新闻',
                    'action' => 'colnews',
                    'controller' => 'news'
                    )
               )
          )
     )
);
```

使用数组的方式创建导航菜单，必须注意数组的嵌套层次以及控制器、方法或链接的URL 的正确性。上面代码创建的菜单效果如图 5.4 所示。

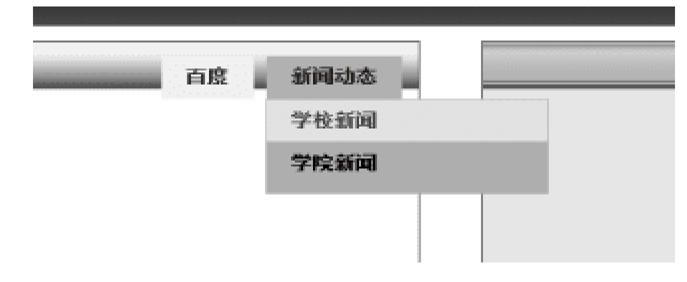


图 5.4　用数组方式创建的导航菜单效果

上面只是简单地介绍了导航菜单的不同创建方式，其实在上面提到的几个类中封装了非常多的方法，使用这些方法可以动态地对导航菜单进行控制，例如增加菜单项、删除菜单项、修改菜单项的属性等，真正做到应用自如。

5.3　新闻资讯页面的实现

系统的设计过程实际上就是系统中信息的存储、处理与显示的过程。信息的存储有多种方式，例如数据库、文件等，应根据系统中信息的具体使用情况进行选用；信息的处理包括信息的增加、删除、修改、查询以及合并等，是对文件或数据库的一系列操作；信息的显示是对信息处理结果的呈现，呈现方式可以是文本、图形以及表格等多种形式。

Zend Framework 项目所采用的 MVC 模式正好契合了系统设计的这个过程。下面以系统中新闻文章页面的实现为例，详细介绍 Zend Framework 中模型的创建，数据库的操作以及页面输出的控制方法。

5.3.1 创建数据库

数据库的创建有多种方式，可以通过命令的方式在 MySQL 控制台中创建，也可以使用 MySQL 的管理工具创建，不管采用什么方式，首先都必须启动 MySQL 数据库服务器。

1. 启动数据库服务

打开 XAMPP 控制面板，单击 MySQL 对应的 Start 按钮，启动 MySQL 数据库服务器。如果控制面板下面的信息显示框中不出现红色的错误信息，则说明数据库服务启动成功，如图 5.5 所示。

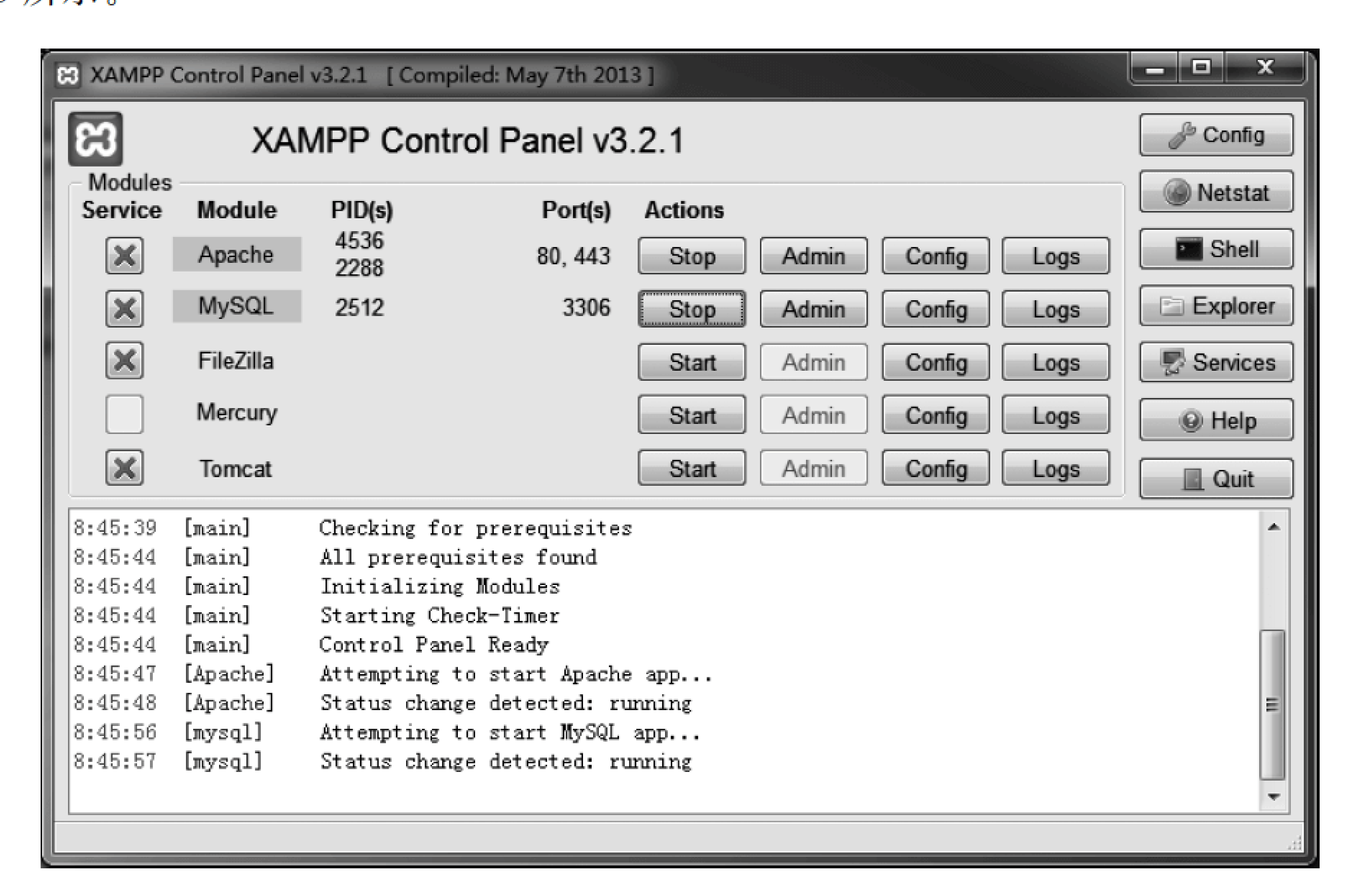


图 5.5 数据库服务器的启动

2. 创建数据库

1）使用 MySQL 控制台

打开操作系统命令窗口，输入以下连接 MySQL 服务器的命令并回车，接着输入密码，即可进入 MySQL 命令提示状态，如图 5.6 所示。

```
mysql -h localhost -u root -p
```

其中，-h 为所连接的数据库服务器位置，可以是 IP 地址，也可以是服务器的域名，这里为 localhost 或 127.0.0.1；-u 为连接数据库服务器使用的用户名，这里为 root；-p 为连接数据库服务器使用的密码。

在如图 5.6 所示的 MySQL 控制台中输入如下命令，创建名为 db_wmoams 的数据库。

```
create database db_wmoams;
```

使用命令：

```
show databases;
```

查看数据库服务器中已创建的数据库，如果在数据库列表中能看到 db_wmoams，说明数据



图 5.6 MySQL 数据库命令窗口

库创建成功。

2）使用 phpMyAdmin 数据库管理工具

依照第 1 章中介绍的方法打开 phpMyAdmin MySQL 数据库管理器，单击窗口左侧的 New 选项或选择窗口右侧的“数据库”菜单项，打开新建数据库窗口，输入数据库名，单击“创建”按钮，即可完成数据库的创建，如图 5.7 所示。



图 5.7 phpMyAdmin 数据库管理器

3. 创建数据表

打开 phpMyAdmin 数据库管理器，找到数据库 db_wmoams，在其中建立名为 tb_news 的数据表，或者用以下命令在 MySQL 控制台中创建。

```
CREATE TABLE IF NOT EXISTS 'tb_news' (
  'id' int(10) unsigned NOT NULL AUTO_INCREMENT,
  'cid' int(10) unsigned NOT NULL DEFAULT '0',
  'uid' int(10) unsigned NOT NULL DEFAULT '0',
  'title' varchar(255) NOT NULL,
  'body' text NOT NULL,
  'type' varchar(50) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL,
  'status' tinyint(4) NOT NULL DEFAULT '1',
  'createtime' int(11) NOT NULL DEFAULT '0',
  'updatetime' int(11) NOT NULL DEFAULT '0',
  'comment' tinyint(4) NOT NULL DEFAULT '0',
  'star' tinyint(4) NOT NULL DEFAULT '0',
  'top' tinyint(4) NOT NULL DEFAULT '0',
  PRIMARY KEY ('id')
) ENGINE = MyISAM  DEFAULT CHARSET = utf8;
```

5.3.2 数据库的配置

在数据库创建好之后，需要对 Zend Framework 框架进行相应配置，让框架能与数据库连接，这样它才能真正地开始为我们服务。

1. 直接修改 application.ini 配置文件

打开 application\configs\application.ini 配置文件，并添加如下代码：

```
…
[development : production]
phpSettings.display_startup_errors = 1
phpSettings.display_errors = 1
resources.frontController.params.displayExceptions = 1

; 数据库
resources.db.adapter = "PDO_MYSQL"
resources.db.params.host = "localhost"
resources.db.params.username = "root"
resources.db.params.password = "wmstudio"
resources.db.params.dbname = "db_wmoams"
resources.db.params.charset = "utf8"
resources.db.isDefaultTableAdapter = true
resources.db.params.driver_options.1002 = "SET NAMES UTF8"
```

代码中的第 1 行声明数据库适配器为 PDO_MYSQL。Zend Framework 提供了一个 PDO 数据库层，可以支持包括 Oracle、MySQL、SQLite、Microsoft SQL Server、PostgreSQL、IBM DB2 等在内的多种数据库。

当使用这些数据库时，只需在配置文件中指定数据库适配器即可，以后如果要更换数据库，也只需更改配置文件，无须改动程序代码。这里使用 MySQL 数据库，所以配置的是 PDO_MYSQL 适配器。如果使用其他类型的数据库，将 PDO_MYSQL 中的 MYSQL 换成相应的数据库类型就可以了。

接下来的 5 行分别配置了数据库服务器(localhost)、使用数据库的用户(root)和密码

(wmstudio)、数据库名称(db_wmoams)以及数据库编码方式(utf8)。

第 6 行指定适配器为项目的默认数据库适配器,这样如果在项目中使用了不止一个数据库,则调用 DefaultTableAdapter 即可方便地使用本数据库。

最后 1 行配置可以使数据在页面、表单间输入与输出都以 utf8 的编码方式进行。需要注意的是,如果添加了这条配置,应确保项目中所有页面的编码方式也是 utf8,以防出现乱码。

2. 使用 ZF Tool 工具配置

打开 Zend Studio 集成开发环境,定位到工作区的项目 wmProject 中,然后选择 Project |Zend Tool 菜单项,打开 Zend Tool 工具命令窗口,输入以下命令:

```
Zf configure db - adapter
"adapter = PDO _ MYSQL&host = localhost&username = root&password = wmstudio&dbname = db _
wmoams&charset = utf8 " development
```

命令行的结尾使用 development 表示数据库的配置放置在 application. ini 文件的 develpment 小节中。

注意以下两行配置无法通过 Zend Tool 工具创建,需要手工添加。

```
resources.db.isDefaultTableAdapter = true
resources.db.params.driver_options.1002 = "SET NAMES UTF8"
```

5.3.3 修改项目命名空间

Zend Framework 中的类名具有特定的命名规则。如果严格按照这种规则命名类,在项目中创建类的实例时就不用通过 include、require 等方式引用类的定义,给编程带来极大的方便。

在 Zend Framework 中搜索某个类的定义时是从 application 目录开始逐步向里进行的,所以,如果类名能够表示该类定义文件的路径,系统就可以自动进行搜索。例如,如果要在项目中定义一个 News 类,该类的定义代码存放在 application\models\News. php 文件中,就可以命名该类为 Application_Model_News。从这里可以看出,应用的默认命名空间前缀为 Application_,这一点也可以在 application\configs\application. ini 中找到相应的定义代码 appnamespace="Application"得到印证。使用这个默认前缀不是很方便,有必要对它进行修改。

更改 Zend Framework 应用程序的命名空间有下面两种方法。

1. 直接修改代码

(1) 打开项目根目录中的. zfproject. xml 文件,找到如下代码:

```
<applicationDirectory classNamePrefix = "Application_">
```

将其修改为:

```
<applicationDirectory classNamePrefix = "Wm_">
```

其中,Wm_是自定义的命名空间前缀。zfproject. xml 文件是 Zend Tool 工具的配置文件,classNamePrefix(类名前缀)预定义了将要创建的控制器、模型、表单等的类名前缀。

(2) 打开 application\configs\application.ini 文件，找到如下代码：

```
appnamespace = "Application"
```

将其修改为：

```
appnamespace = "Wm_"
```

2. 用 ZF Tool 命令工具修改

在 Zend Studio 集成开发环境中打开 Zend Tool 工具命令窗口，输入命令：

```
Zf change application.class - name - prefix Wm
```

即可自动完成上述 1 中的两处代码修改。

5.3.4 创建模型与方法

对于 Zend Framework 的 MVC 模式，我们已经用到了 V 和 C，即视图与控制器。下面针对系统中的“新闻文章”这类信息编写处理它们的业务逻辑（即模型）来实现对信息的一系列操作。

1. 创建模型

打开 Zend Studio 集成开发环境，定位到工作区的项目 wmProject 中，选择 Project | Zend Tool 菜单项，打开 Zend Tool 工具命令窗口，输入命令：

```
Zf create model News
```

创建一个名为 News 的模型。若命令执行成功，Zend Tool 工具会在项目的 application 目录下新建一个名为 models 的文件夹，并在这个文件夹中创建一个名为 News.php 的文件，该文件定义了一个名为 Wm_Model_News 的类。为了使用 Zend Framework 的 Zend_Db 组件对数据库进行操作，需要给这个类添加一个名为 Zend_Db_Table 的基类，并添加对应的数据库表名。代码如下：

```
class Wm_Model_News extends Zend_Db_Table
{
    //定义模型数据表
    protected $_name = 'tb_news';

}
```

模型类 Wm_Model_News 继承 Zend_Db_Table 类后，便成为 Zend Framework 的表模型，拥有了对数据库操作的权限。代码中的变量_name 为表模型中的特定变量，代表该表模型对应的数据表。

2. 创建方法

为 Wm_Model_News 类添加一个名为 getNews 的方法，用于从数据库中读取一条新闻信息。代码如下：

```
class Wm_Model_News extends Zend_Db_Table
{
    protected $_name = 'tb_news';
```

```
    public function getNews( $ where = null, $ order = null){
        if(is_numeric( $ where)){
            $ row = $ this -> find( $ where) -> current();
        }else{
            $ row = $ this -> fetchRow( $ where, $ order);
        }
        if ( $ row) {
            return $ row;
        }else {
            return null;
        }
    }
```

5.3.5 实现新闻文章的显示

1. 创建控制器

打开 Zend Studio 集成开发环境，定位到工作区的项目 wmProject 中，选择 Project | Zend Tool 菜单项，打开 ZF Tool 工具命令窗口，输入命令：

```
Zf create controller News
```

在默认模块 default 中创建一个名为 News 的控制器。在 News 控制器的 index 方法中添加以下代码：

```
class NewsController extends Zend_Controller_Action
{
    public function init()
    {
        /* Initialize action controller here */
    }
    public function indexAction()
    {
        $ modelNews = new Wm_Model_News();            //实例化模型对象
        $ where = array('top' => 1);                  //定义查询条件
        $ newsTop = $ modelNews -> getNews( $ where);  //获取新闻文章
        $ this -> view -> newsTop = $ newsTop;         //输出到视图
    }
}
```

上面的代码中设置的查询条件是“置顶于新闻文章的第 1 篇”。

2. 修改视图

打开 News 控制器的 index 方法对应的视图文件 application\views\scripts\news\index.phtml，将其修改为：

```
<br /><br />
<?php
    if( $ this -> newsTop){
        echo "<h3>". $ this -> newsTop['title']."</h3>";
        echo "发布日期: ".date("Y-m-d : H-i-s", $ this -> newsTop['createtime']).
"<hr />";
        echo $ this -> newsTop['body'];
    }else
        echo "<h3>抱歉,没有搜索到您要查找的新闻文章!<h3><hr />";
?>
```

3. 显示新闻

用 phpMyAdmin 数据库管理器打开数据库 db_wmoams，在数据表 tb_new 中添加一篇新闻文章，设置 top 为 1，表示该篇文章需要置顶显示。

在浏览器地址栏中输入 http://wmoams.com/news，页面效果如图 5.8 所示。

大家可以在数据表中添加不同的新闻演示数据，并修改 News 控制器的 index 方法中的 where 查询条件（例如输入数字、字符串或有多个查询条件的数组）测试页面的显示效果。



图 5.8 新闻文章的显示

5.4 新闻的列表及详细显示

5.3 节通过一篇新闻文章的显示介绍了 MySQL 数据库的设计、MVC 模式中模型的构建方法以及通过控制器使用模型，然后输出到视图的全过程。相信大家对 Zend Framework 框架中模型(M)、视图(V)、控制器(C)三者之间的关系应该有了更进一步的了解。下面对模型的功能进行完善与扩充，使其能从数据库中查询到多篇文章，并以列表的形式在视图中显示出来。

5.4.1 新闻的列表显示

1. 添加模型方法

前面在 Wm_Model_News 模型中添加 getNews 方法，实现了对某类新闻文章的查询。下面继续在该模型中添加方法，使其能一次查询到多篇新闻文章。

启动 Zend Studio 集成开发环境，打开 wmProject 项目中的 application\models\News.php 模型文件，添加 getNewslist()方法，代码如下：

```
public function getNewslist( $ where = null, $ order = null, $ limit = null)
{
        $ rowSet = $ this -> fetchAll( $ where, $ order, $ limit);
        if ( $ rowSet -> count()> 0) {
            return $ rowSet;
        }else {
            return null;
        }
    }
```

2. 编写控制器代码

打开 News 控制器文件 application\controllers\NewsController. php，在 index()方法中添加代码。

```
public function indexAction()
{
        $ modelNews = new Wm_Model_News();              //实例化模型对象
        $ where = array( 'top' = > 1);                  //定义查询条件
        $ newsTop = $ modelNews -> getNews( $ where);   //获取新闻文章
        $ this -> view -> newsTop = $ newsTop;          //输出到视图

        $ where = array( 'top' = > 0);
        $ order = 'createtime DESC';
        $ limit = 3;
        $ newsList = $ modelNews -> getNewslist( $ where, $ order, $ limit);
        $ this -> view -> newsList = $ newsList;        //输出到视图
    }
```

3. 显示新闻列表

在 News 控制器的 index()方法中调用 Wm_Model_News 模型类的 getNewslist()方法获取满足设定条件的新闻文章列表，这个列表是一个 Zend_Db_Table_Rowset 类对象，它是数据库记录的集合，也就是说该对象中又包含了许多 Zend_Db_Table_Row 类的子对象。为了输出这些子对象中的键值，需要采用循环的方式输出。

打开视图文件 application\views\scripts\news\index. phtml，添加以下代码：

```
< br />< br />
<?php
    if( $ this -> newsTop){
    echo "< h3 >". $ this -> newsTop[ 'title']. "</h3 >";
    echo "发布日期：". date("Y - m - d : H - i - s", $ this -> newsTop[ 'createtime']).
"< hr />";
    echo $ this -> newsTop[ 'body']. "< hr />< br />";
    }else
    echo "< h3 >抱歉,没有搜索到您要查找的新闻文章!< h3 >< hr />";
    echo "< h3 >其他新闻</h3 >< hr />";
    echo "< ul >"
    if( $ this -> newsList){
        foreach ( $ this -> newsList as $ value){
            echo "< li >". $ value[ 'title']. "< l li >";
        }
    echo "</ul >";
```

```
    }
?>
```

打开浏览器进行测试，效果如图 5.9 所示。



图 5.9　新闻文章的列表显示

4. 新闻列表显示方式的改善

使用 Zend_View 的 partialLoop 视图助手代替视图中的 foreach 输出循环能够减少视图页面中的 PHP 代码，更能体现面向对象的设计理念。

修改视图文件 application\views\scripts\news\index.phtml 中的代码：

```
<?php
    …
    echo "<h3>其他新闻</h3><hr />";
    if( $this->newsList){
        echo "<ul>";
        echo $this->partialLoop('partials/row-pages.phtml', $this->newsList);
        echo "</ul>";
    }
?>
```

在 application\views\scripts 目录下新建 partials 文件夹，在其中新建 row-pages.phtml 文件，添加代码：

```
<li>
    <a href="#"><?php echo $this->title; ?></a>
    <span>发布时间：
        <?php echo date('Y-m-d : H-i-s', $this->createtime); ?>
  </span>
  </li>
```

在浏览器中测试,效果如图 5.10 所示。



图 5.10　新闻文章列表显示的完善

5.4.2　新闻的详细显示

在新闻首页,文章标题应是一个超级链接,单击后可以进入文章详细页面,显示文章的详细信息,如图 5.10 所示。图中所展示的新闻首页分为两部分,上部分为置顶显示的重要新闻,下部分是其他新闻的列表,我们给每个新闻标题设置了超级链接,接下来实现单击标题,进入新闻文章详细页面的功能。

1. 在新闻首页中添加链接

新闻列表中的每条标题都是一个超级链接,因此,通过单击标题就可以显示该篇新闻的详细内容。为了在数据库中准确地查询到该条新闻记录,需要通过 get 方法将新闻的序号传递给模型。

打开 application\views\scripts\partials\row-pages.phtml 文件,给 a 标签的 href 属性赋值:

```
<li>
    <a href="/news/detail/id/<?php echo $this->id ?>">
        <?php echo $this->title; ?>
    </a>
    <span>发布时间:
            <?php echo date('Y-m-d : H-i-s', $this->createtime); ?>
    </span>
    </li>
```

代码中的 news、detail 分别表示控制器与方法,id 是我们自定义用来传递新闻序号的参数,它的值会通过 get 方法传递到 detail 方法中。

2. 创建控制器方法

打开 Zend Studio 集成开发环境,定位到工作区的项目 wmProject 中,选择 Project|

Zend Tool 菜单项，打开 Zend Tool 工具命令窗口，输入命令：

```
Zf create action detail News
```

在 News 控制器中创建 detail 方法。打开 application\controllers\NewsController.php 文件，在 detailAction 方法中编写代码：

```
public function detailAction()
{
      $ id = $ this -> getRequest() -> getParam('id');
      $ modelNews = new Wm_Model_News();
      $ news = $ modelNews -> getNews( $ id);
      $ this -> view -> news = $ news;
 }
```

代码中的第 1 行获取 news\index.phtml 视图通过 get 方法传递的 URL 参数 id 的值，然后实例化一个新闻模型，调用模型中的 getNews 方法，从数据库中得到该新闻的记录，最后将得到的记录传递到视图。

3. 页面视图设计

在创建 detail 方法时，Zend Framework 同时创建了对应的视图 detail.phtml，它位于 application\views\scripts\news 目录下。打开该文件，将代码改为：

```
< br />< br />
<?php
echo "< h2 >". $ this -> news -> title."</h2 >";
echo "< span >发表于: ".date('Y - m - d', $ this -> news -> createtime)."</span >";
echo "< span >更新于: ".date('Y - m - d', $ this -> news -> updatetime)."</span >";
echo "< hr />";
echo  $ this -> news -> body;
```

完成上述工作后，在浏览器中输入 http://wmoams.com/news 访问新闻主页，然后在新闻列表中单击某个标题，即可看到该新闻的详细内容，如图 5.11 所示。



图 5.11　新闻的详细显示

5.5 Zend_Db 组件

Zend_Db 是 Zend Framework 的数据库操作组件，它包括 Zend_Db_Adapter、Zend_Db_Statement、Zend_Db_Profiler、Zend_Db_Select、Zend_Db_Table、Zend_Db_Table_Row 以及 Zend_Db_Table_Rowset 等几个部分。

5.5.1 Zend_Db_Adapter 类

Zend_Db_Adapter 类是 Zend Framework 的数据库抽象层 API，基于 PDO(PHP Data Object，PHP 数据对象扩展)，可以使用它连接和处理多种数据库，例如 Microsoft SQL Server、MySQL 以及 SQLite 等。

1. 创建适配器对象

用户可以通过 Zend_Db 类的静态方法 factory()创建 Zend_Db_Adapter 对象。该方法的语法格式如下：

```
Zend_Db::factory( $adapter, $config);
```

其中，参数 adapter 是一个字符串变量，表示 Zend_Db_Adapter 支持的数据库类型，例如 DB2、MySQL、Oracle 数据库等，以及支持使用 PDO 的类型，例如 PDO_Ibm、PDO_Mssql、PDO_Mysql、PDO_Sqlite 等；参数 config 为一个数组型变量，该数组为连接一个数据库必须配置的内容，就像前面在 application\configs\application.ini 文件中配置的那样。例如，下面的代码实现与本地 MySQL 数据库 db_test 的连接。

```
$options = array(
        'host' =>'localhost',
        'username' =>'root',
        'password' =>'123456',
        'dbname' =>'db_test'
 );
 $db = Zend_Db::factory('PDO_MYSQL', $options);
```

2. 查询记录

在创建了一个适配器对象之后，就可以通过它对数据库进行查询了。查询语句可以直接在 Zend_Db_Adapter 类的 query 方法中编写，也可以采用 Zend_Db_Select 对象。例如：

```
$db = Zend_Db::factory('PDO_MYSQL', $option);
$result = $db->query('select * from tb_user');
$resultset = $result->fetchAll();
```

或者

```
$db = Zend_Db::factory('PDO_MYSQL', $option);
$select = $db->select();
$select->from('tb_use','*');
$resultset = $db->fetchAll( $select);
```

实际使用时，需要对查询语句进行无害化处理，即对语句中的引号进行加斜线处理，以

防止数据库的 SQL 注入攻击，此时使用 Zend_Db_Adapter 类的 quote 或 quoteInto 方法。

例如：

```
$ value = $ db -> quote("John's");
```

$value 的值已变成：

```
'John\'s'
```

又如：

```
$ result = $ db -> quoteInto('select * from tb_test where id < ?', '5');
```

用户提供一个包含问号占位符的基础字符串，然后在该位置加入带引号的标量或者数组。在使用 quoteInto 方法后，返回一个安全的 SQL 字符串。例如：

```
select * from db_test where id < '5'
```

3. 插入记录

使用 Zend_Db_Adapter 类的 insert 方法实现数据库记录的插入。为了方便，常常使用 insert 方法将要插入的数据绑定，并创建一个 insert 语句。因为绑定的数据是自动进行加引号处理的，这样可以有效避免对数据库的攻击。其语法格式如下：

```
insert( $ table,array $ bind)
```

使用示例：

```
$ user = array{
       'username' => 'weimeng',
       'password' => '123456'
}
$ table = tb_user;
$ insert = $ db -> insert( $ table, $ user);
```

注意：insert 方法的返回值并不是最后插入记录的 ID，因为在一些表中并没有一个自增的字段。该方法的返回值是改变记录的行数，通常为 1。

4. 修改记录

通过执行 Zend_Db_Adapter 类的 update 方法实现数据库中记录的修改。其语法格式如下：

```
update( $ table,array $ bind, $ where = '')
```

其中，参数 table 为表名；bind 为修改字段与内容构成的绑定数组；where 为条件语句。示例代码如下：

```
$ table = tb_user;
$ user = array{
       'password' => 'abcdef'
};
$ where = 'username = weimeng';
$ db -> update( $ table, $ user, $ where);
```

5. 删除记录

通过执行 Zend_Db_Adapter 类的 delete 方法实现数据库中记录的删除。其语法格式如下：

```
delete( $ table, $ where = '')
```

其中，参数 table 为表名；where 为删除条件。

5.5.2 Zend_Db_Table 类

Zend_Db_Table 类是 Zend Framework 的表模块，该模块通过 Zend_Db_Adapter 连接到数据库，为数据库模式检查表对象，并对该表执行查询、添加、修改以及删除等操作。

Zend_Db_Table 类为抽象类，所以不能直接对该类实例化，只能先继承该类然后实例化子类。首先需要设定一个默认对数据库的适配器，如果不再指定其他类型数据库适配器，则所有的 Zend_Db_Table 类实例都会使用默认的适配器。

例如，本章案例中的 Wm_Model_News 模型就是 Zend_Db_Table 类的子类，它的默认适配器是在配置文件 application\configs\application. in 中以资源的形式定义的。当在控制器的方法中实例化该模型时，会调用该类的基类 Zend_Db_Table_Abstract 的构造方法及 _setup 方法初始化数据库适配器。

1. 表名及主键的设置

在默认情况下，Zend_Db_Table 类会将其类名作为数据库的表名(大小写不同的地方用下划线分隔)。例如，一个名为 MyTableUser 的 Zend_Db_Table 类将自动对应数据库中的 my_table_user 数据表。若要修改数据表名，将该类的_name 属性重新设置即可，就像我们在 Wm_Model_News 类中做的那样。

Zend_Db_Table 类的默认主键为 id 字段，若要重新设置，修改其属性_primary。例如，若要将案例新闻表中的 createtime 字段设置为类的查询主键，添加如下代码：

```
class Wm_Model_News extends Zend_Db_Table
{
    //定义模型数据表
    protected $ _name = 'tb_news';
    protected $ _primary = 'createtime';
}
```

2. 通过主键查询记录

通过调用 Zend_Db_Table 类的 find 方法可以直接使用主键值查询表中记录。例如 Wm_Model_News 类的方法中的代码：

```
public function getNews( $ where = null, $ order = null){

        if(is_numeric( $ where)){
            $ row = $ this - > find( $ where) - > current();
        }else{
            $ row = $ this - > fetchRow( $ where, $ order);
        }
…
}
```

Zend_Db_Table 类的 find 方法返回一个 Zend_Db_Table_Rowset 对象，输出时要进行相应的转换。另外，带入 find 方法中的参数可以是数值，此时查询一条记录；也可以是数组，例如 array(1,5,10)，表示查询主键为 1、5、10 的记录，共 3 条。

3. 通过条件查询记录

虽然使用 find 方法通过主键能轻松查询到数据库记录，但更多时候需要通过条件查找。Zend_Db_Table 类提供了方法 fetchRow 和 fetchAll 实现条件查找功能。

使用 fetchRow 方法查找一条记录，返回一个 Zend_Db_Table_Row 类的对象。例如：

```
public function getNews( $ where = null, $ order = null){

        if(is_numeric( $ where)){
            $ row = $ this -> find( $ where) -> current();
        }else{
            $ row = $ this -> fetchRow( $ where, $ order);
        }
…
}
```

使用 fetchAll 方法查找多条记录，返回一个 Zend_Db_Table_Rowset 类的对象。例如：

```
public function getNewslist( $ where = null, $ order = null, $ limit = null){
  $ rowSet = $ this -> fetchAll( $ where, $ order, $ limit);
      if ( $ rowSet -> count()> 0) {
          return $ rowSet;
      }else {
          return null;
      }
  }
```

5.5.3 Zend_Db_Select 类

Zend_Db_Select 类是一种不受数据库约束构建 select 的 SQL 语句的工具，可使用该组件生成查询的 SQL 语句，而不需要考虑各种数据库 SQL 语句的差别，并可以有效避免对 SQL 语句的攻击。

1. 创建实例

创建一个 Zend_Db_Select 类的实例，使用 Zend_Db_Adapter 的 select 方法。调用该方法不需要任何参数就可以返回一个 Zend_Db_Select 实例对象，例如：

```
$ db = Zend_Db::factory('PDO_MYSQL', $ option);
$ select = $ db -> select();
```

2. 基本查询

指定查询表名及条件后，就可以通过 fetchAll 执行查询了。例如：

```
$ db = Zend_Db::factory('PDO_MYSQL', $ option);
$ select = $ db -> select();
$ select -> from('tb_use', '*');
$ resultset = $ db -> fetchAll( $ select);
```

上面的代码相当于执行了：

```
select * from tb_use
```


查询语句。

3. 条件查询

在基本查询的基础上，通过 where 或 orWhere 方法设置查询条件即可。例如：

```
$ db = Zend_Db::factory('PDO_MYSQL', $ option);
$ select = $ db -> select();
$ select -> from('tb_use','*');
$ select -> where('id < 5');
$ resultset = $ db -> fetchAll($ select);
```

4. 查询结果的排序

使用 Zend_Db_Select 类的 order 方法对查询返回结果集按照给定的排序规则进行排序。例如：

```
$ db = Zend_Db::factory('PDO_MYSQL', $ option);
$ select = $ db -> select();
$ select -> from('tb_use','*');
$ select -> where('id < 5');
$ select -> order('username ASC');
$ resultset = $ db -> fetchAll($ select);
```

5. 限制查询结果数

在使用数据库查询的结果集时，常常需要限制结果集的数量，这个功能通过调用 Zend_Db_Select 类的 limit 方法实现。例如：

```
$ db = Zend_Db::factory('PDO_MYSQL', $ option);
$ select = $ db -> select();
$ select -> from('tb_use','*');
$ select -> where('id < 5');
$ select -> order('username ASC');
$ select -> limit(2,3);
$ resultset = $ db -> fetchAll($ select);
```

在上面的 limit 方法中，第 1 个参数“2”表示取两条记录；第 2 个参数“3”表示从结果集的第 3 条记录开始取。因此，上面的查询执行后的结果是获取到了表 tb_user 中序号小于 5 的所有记录中的第 3、4 条记录。

5.6 本章小结

本章主要介绍 Zend Framework 应用的导航及数据库操作功能。Zend Framework 的导航由 Zend_Navigation 组件完成，通过该组件不仅可以实现多级导航菜单，还可以实现页面的面包屑导航功能。但使用 Zend_Navigation 组件稍显复杂，大家可以根据实际情况选用其他合适的解决方案，例如 CSS、jQuery 等。

数据库操作是 Web 应用开发的关键，当然也是本章的重点。Zend Framework 应用的

数据库操作由 Zend_Db 组件实现，该组件提供的类模块功能强大、兼容性好，能够充分满足各种业务逻辑的需要。本章通过新闻资讯页面的显示实例循序渐进、由浅入深、详细地讲解了 Zend Framework 对数据库的操作方法，包括数据库的配置、模型的创建以及查询数据在页面中的显示等。在介绍新知识的过程中，我们始终围绕 MVC 模式，进一步阐述了 Zend Framework 框架中的 M(模型)、V(视图)、C(控制器)之间的合作关系，用于加深并巩固前面章节中所学的知识。

第6章 注册登录及 Zend_Form 表单

表单是 Web 应用与用户交互的窗口，同时也是应用受到外来攻击的入口。因此，表单的设计在 Web 应用中显得尤为重要，关系到用户的体验、数据的准确以及网络安全等各个方面。

本章通过用户注册与登录功能的实现介绍 Zend Framework 的 Zend_Form 表单的创建、装饰及使用方法。

6.1 登录表单设计

在第 4 章中已经创建好了登录页面，但没有添加登录表单，下面通过创建 Zend Framework 表单来实现系统的登录功能。

6.1.1 登录页面效果

启动 Apache 服务器，在浏览器中输入 http://wmoams.com，进入系统首页。单击页面中的教材名称图片，即可进入系统用户登录页面，如图 6.1 所示。



图 6.1 用户登录页面效果

如果在图 6.1 所示的登录页面中直接单击“登录”按钮，则会出现如图 6.2 所示的错误提示页面。



图 6.2　用户登录错误信息

可以看到，由于在登录表单中没有填写任何信息，页面右侧出现了错误提示框。提示框中的错误信息是在登录表单类中设置的，图 6.2 中列出的是所有错误提示信息。

按图 6.2 提示中对登录名的要求输入正确的登录名，然后单击“登录”按钮，出现如图 6.3 所示的密码输入错误信息。



图 6.3　密码输入错误信息

这里给出的是对两个密码输入框中的数据进行验证后的错误提示信息，若表单数据不能通过验证，则返回登录页面。这些功能都是使用 Zend Framework 的 Zend_Form 表单自动完成的。

6.1.2　Zend_Form 表单的创建

1. 创建 Zend_Form 表单类

启动 Zend Studio 集成开发环境，选择工作区中的 wmProject 项目，打开 ZF Tool 工具

命令窗口,输入命令:

```
Zf create form Login
```

创建一个名为 Login 的 Zend_Form 表单。

以上命令自动创建 application\forms\Login. php 表单文件,其中的 forms 文件夹也是 ZF Tool 工具自动创建的。打开文件会看到表单类定义的一些初始代码,如下所示:

```
class Wm_Form_Login extends Zend_Form
{
    public function init()
    {
        /* Form Elements & Other Definitions Here ... */
    }
}
```

从上面的代码可知,创建的表单类名为 Wm_Form_Login,该名称符合前面所讲的 Zend Framework 类名的命名规则,使用该类时可以自动加载。另外,表单类继承于 Zend_Form 类,拥有该类的全部功能。Zend_Form 类位于 Zend Framework 库文件的 Zend 目录下,它的其他组件类位于 Zend\Form 目录下,大家可以打开源文件了解其功能。

2. 添加表单元素

在表单类 Wm_Form_Login 的 init 方法中添加代码,创建表单元素。为了简单,这里只创建"用户"、"密码"和"提交"3 个表单元素,其他(如"验证码"等)元素在以后的章节中介绍。代码如下:

```
class Wm_Form_Login extends Zend_Form
{
    public function init()
    {
        //表单属性
        $this->setName('form_login')
             ->setMethod('post');
        //用户名输入文本框
        $userName = $this->createElement('text','username');
        $userName->setLabel('用户名: ');
        $userName->setRequired(TRUE);
        $userName->addValidator('stringLength',false,array(5,20));
        $userName->addErrorMessage('用户名要求英文 5-20 个字母或 2-6 个汉字');
        $this->addElement($userName);
        //密码输入框
        $password = $this->createElement('password','password');
        $password->setLabel('密码: ');
        $password->setRequired(TRUE);
        $password->addValidator('stringLength',false,array(6));
        $password->addErrorMessage('密码要求至少 6 个字符');
        $this->addElement($password);
        //提交按钮
        $submit = $this->createElement('submit','提交');
        $this->addElement($submit);
    }
}
```

由于 Wm_Form_Login 表单继承于 Zend_Form 类，打开 Zend Framework 使用手册或 Zend Framework 库文件中的 Zend_Form 类源码，可以查询其全部方法。

在上述代码中，调用 Zend_Form 类的 setName、setMethod、createElement、addElement 方法对表单对象的名称、数据提交方法进行设置，并完成创建与添加表单元素的操作。

代码中的变量 userName、password 和 submit 分别为“文本框”、“密码框”和“提交按钮”对象，它们分别属于 Zend_Form_Element_Text、Zend_Form_Element_Password 和 Zend_Form_Element_Submit 类的表单元素对象。方法 setLabel、setRequired、addValidator 和 addErrorMessage 为表单元素对象的方法，完成表单元素的属性、验证器以及验证错误信息的设置。对于表单元素方法的使用请参考相应类的定义或 Zend Framework 手册。

Zend_Form 表单支持的表单元素，除了上面 3 种以外还有很多，在接下来的 6.2 节中将详细介绍。

3. 显示表单

打开控制器文件 application\controllers\User.php，在 User 控制器的 login 方法中添加如下代码：

```
class UserController extends Zend_Controller_Action
{
    …
    public function loginAction()
    {
        …
        $formLogin = new Wm_Form_Login();
        $this->view->formLogin = $formLogin;
    }
}
```

在 login 方法中，首先创建一个 Wm_Form_Login 类的对象，然后将该对象赋给视图对象中的 formLogin 变量，formLogin 是自定义变量。

从前面的分析可知，控制器的 login 方法对应的视图文件是 application\views\scripts\user\login.phtml 文件，因此，只需要在该视图对象中把表单对象输出即可显示表单。

打开 User 控制器的 login 方法对应的视图文件 login.phtml，在登录区的右侧 div 标签中添加如下代码：

```
…
<div id="login_center">
  <div class="left">
      <img src="<?php echo '/images/login_center_mark.gif'; ?>" />
  </div>
  <div class="center"></div  >
  <div class="right"><?php echo $this->formLogin; ?></div>
</div>
…
```

上述阴影部分的代码就是用来显示登录表单的。其中的对象 formLogin 是在 loginAction 中通过语句：

```
$ this - > view - > formLogin = $ formLogin;
```

传递到视图中的表单对象。注意,在视图中 this 后面没有 view,因为这时的 this 表示的就是视图对象本身。

在浏览器中输入 http://wmoams.com/user/login,页面效果如图 6.4 所示。



图 6.4 登录表单页面

从图 6.4 可以看出,登录表单对象正常显示,但布局格式不尽如人意。打开该页面在浏览器中的源文件:

```
< div class = "right">
< form id = "form_login" enctype = "application/x - www - form - urlencoded" method = "post"
action = "">
< dl class = "zend_form">
< dt id = "username - label">< label for = "username" class = "required">用户名: </label ></dt >
< dd id = "username - element">
< input type = "text" name = "username" id = "username" value = "" /></dd >
< dt id = "password - label">< label for = "password" class = "required">密码: </label ></dt >
< dd id = "password - element">
< input type = "password" name = "password" id = "password" value = "" /></dd >
< dt id = "提交 - label"> &# 160;</dt >< dd id = "提交 - element">
< input type = "submit" name = "提交" id = "提交" value = "提交" /></dd >
</dl >
</form >
</div >
```

从源代码中可以看出,Zend_Form 表单在页面中输出时使用了<dl>、<dd>、<dt>标签,并且使用了样式类 zend_form、required 等对布局进行控制,因此可以通过对这些标签或样式类进行重新定义来满足自己的特定布局要求。

例如,在 login. phtml 中添加如下样式代码:

```
< style type = "text/css">
  dd{float:left;display:inline;height:30px; margin:15px}
  dt{float:left;display:inline;height:30px;margin:15px}
  # password - element{margin - left:25px}
  .zend_form{width:300px;height:200px}
</style >
```

刷新浏览器中的登录界面,效果如图 6.5 所示。



图 6.5　修改登录页面样式后的效果

图 6.5 所示的效果图清楚地说明,通过重写样式文件可以有效地控制表单的布局方式。这种设置页面布局的方式在实际应用中不太方便,当有多个表单对象存在时尤其混乱。

Zend Framework 框架提供了一种叫"表单装饰器"的布局管理方法,能够非常灵活地对 Zend_Form 表单进行布局设置。对于"表单装饰器"将在本章的 6.3 节进行介绍。

4. 表单的处理

在表单创建完毕并正常显示后,接下来就是对表单数据的处理。用户在登录表单中输入"用户名"、"密码"后,数据被提交到 User 控制器的 login 方法中。数据的处理包括数据的接收、有效性验证、注入安全防范、与数据库中的注册用户比对、记录登录时间等一系列操作。这里只介绍表单的简单处理过程,假设用户名、密码分别为 wmstudio 和"123456"的用户为合法用户。

在 User 控制器的 login 方法中添加如下代码:

```
class UserController extends Zend_Controller_Action
{
    …
    public function loginAction()
    {
        …
        $formLogin = new wm_Form_Login();
         if($this->getRequest()->isPost()){
             if($formLogin->isValid($_POST)){
                 $data = $formLogin->getValues();
                 if($data['username'] == "wmstudio"
&& $data['password'] == "123456")
                     $this->redirect('/index/main');
                 else
                     $this->view->message = "用户名或密码错误!";
             }
        }
        $this->view->formLogin = $formLogin;
    }
}
```

在上述代码中,首先判断请求是否来自于表单的提交,即是否为用户单击表单中的"提交"按钮后的请求;若是,则进入登录处理程序。在登录处理模块中,先进行表单数据的有效性验证,这些有效性规则是在表单类的定义中设置的,例如前面设置的"用户名要求英文 5-20 个字母或 2-6 个汉字"、"密码要求至少 6 个字符"等。如果用户输入的数据符合设定的

规则,便开始用户名与密码的验证。这里只进行简单的登录流程验证,更多细节将在 6.5 节进行介绍。

在浏览器中输入 http://wmoams.com/user/login,打开用户登录页面。在表单中输入不同的数据,页面效果如图 6.6 和图 6.7 所示。



图 6.6　验证错误页面效果



图 6.7　非法用户登录页面效果

从图 6.6 可知,用户输入字符的长度少于设定的长度时,验证器工作正常,正确给出了错误提示信息。图 6.7 所示为用户输入的用户名或密码与注册用户比对不符合,控制器也正确地给出了错误信息。

注意:此时错误信息的显示是以列表的形式分别显示在各表单元素下面的。

6.2　Zend_Form 表单

上面通过实例操作了解了 Zend_Form 表单的创建、简单修饰与处理的基本过程。Zend_Form 表单在 Web 应用中能简化表单的处理,尤其是能提供元素数据的过滤与校验。它充分利用 Zend Framework 的其他组件(例如 Zend_Config、Zend_Validate、Zend_Filter、Zend_Loader_PluginLoader 以及 Zend_View 等)协同实现其强大的功能。

6.2.1　Zend_Form 表单元素

1. 表单元素类型

Zend Framework 的具体表单元素类涵盖了大部分的 HTML 表单元素,常用的类型及

对应的类见表 6.1。

表 6.1　Zend_Form 表单元素及类

类　　型	类
button	Zend_Form_Element_Button
checkbox	Zend_Form_Element_Checkbox
hidden	Zend_Form_Element_Hidden
image	Zend_Form_Element_Image
password	Zend_Form_Element_Password
radio	Zend_Form_Element_Radio
reset	Zend_Form_Element_Reset
select	Zend_Form_Element_Select
submit	Zend_Form_Element_Submit
text	Zend_Form_Element_Text
textarea	Zend_Form_Element_Textarea

2. 表单元素的创建

1) 使用 Zend_Form 对象的 createElement 方法

在 6.1 节登录表单元素的创建过程中，我们使用的就是这种方法。使用该方法创建表单元素分为 3 个步骤：首先创建表单元素对象；然后为元素对象属性赋值或添加校验器；最后将表单元素添加到表单对象中。

createElement 方法属于 Zend_Form 类，其原型如下：

```
public function createElement($type, $name, $options = null);
```

从函数的形参列表可以看出，调用该函数需要带两个或 3 个参数。第 1 个参数 type 表示表单元素的类型，即表 6.1 中的类型名称，如登录表单中的用户名文本输入框和密码输入框的类型分别为 text 和 password；第 2 个参数是元素对象的 name 属性值，这是表单数据传递时的变量名称，如前面登录框中的 username 与 password；第 3 个参数是一个默认参数，前面采用的是它的默认形式。通过这个参数可以直接定义表单元素的样式，例如登录框中“用户名”文本输入框的创建，也可以写成如下形式：

```
$userName = $this->createElement('text','username',array(
            'label'=>'用户名：',
                'required'=>true,
                'validator'=>array('stringLength',false,array(5,20)),
                'erroMessage'=>'用户名要求英文 5-20 个字母或 2-6 个汉字'
        ));
```

代码中的 required 表示该表单元素是必要的，stringLength 是一个字符长度校验器。当然，还可以添加其他类型的核验器及过滤器。

2) 使用表单元素对象创建

用户可以不使用 Zend_Form 对象的 createElement 方法，直接用 new 创建所需的表单元素对象，并对表单对象属性赋值或配置校验器与过滤器，然后用 Zend_Form 对象的 addElement 方法将表单元素添加到表单对象中。代码如下：

```
$ this -> addElement(new Zend_Form_Element_Text('username',
        array(  'label' => '用户名：',
                'required' => true,
                'validator' => array('stringLength',false,array(5,20)),
                'erroMessage' => '用户名要求英文 5 - 20 个字母或 2 - 6 个汉字'
        )));
```

6.2.2 Zend_Form 表单属性设置

在 HTML 中，表单具有多个属性，例如 name、id、method、action 等。Zend_Form 表单在创建时也要对这些属性进行设置。

在 Zend Framework 库中找到文件 Form. php，打开 Zend_Form 类的源文件；或在 Zend Studio 集成开发环境中用鼠标指向代码中的 Zend_Form 类名，单击智能指示框中的显示源文件图标，快速打开 Zend_Form 类的源文件，如图 6.8 所示。

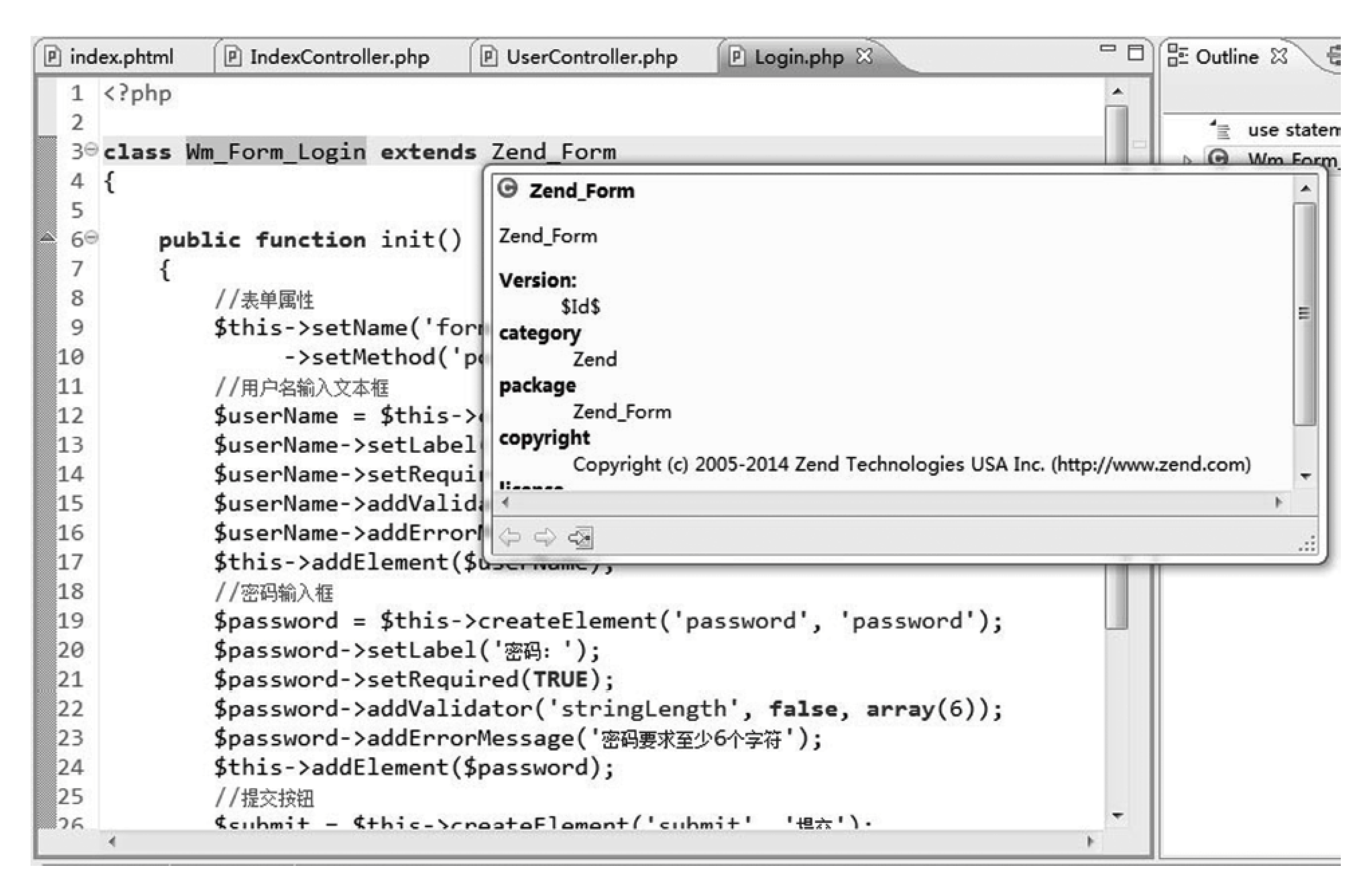


图 6.8 Zend Studio 集成开发环境的智能提示

Zend_Form 类中有很多带有 set 前缀的方法，如图 6.9 所示，使用这些方法对表单属性进行设置。

在前面用户登录表单的创建中，我们使用了 setName 方法设置表单的 name 属性值为 form_login；使用 setMethod 方法设置表单的 method 属性值为 post。除此之外，还可以使用 setAction、setAttrib、setAttribs 等方法设置表单的 id、action、style、class 等属性值。代码如下：

```
$ this -> setName('form_login')
     -> setMethod('post')
     -> setAction('/user/register')
     -> setAttrib('id','login_form')
```

```
->setAttrib('class','form_login_class')
->setAttrib('style','margin:0px;padding:0px');
```

在浏览器中显示的上述表单的源代码如下：

```
<form id="login_form" enctype="application/x-www-form-urlencoded"
          method="post"
         action="/user/registry"
         class="form_login_class"
         style="margin:0px;padding:0px">
```

从代码可以看出，在表单类中设置的表单属性得到了浏览器的正确解析。

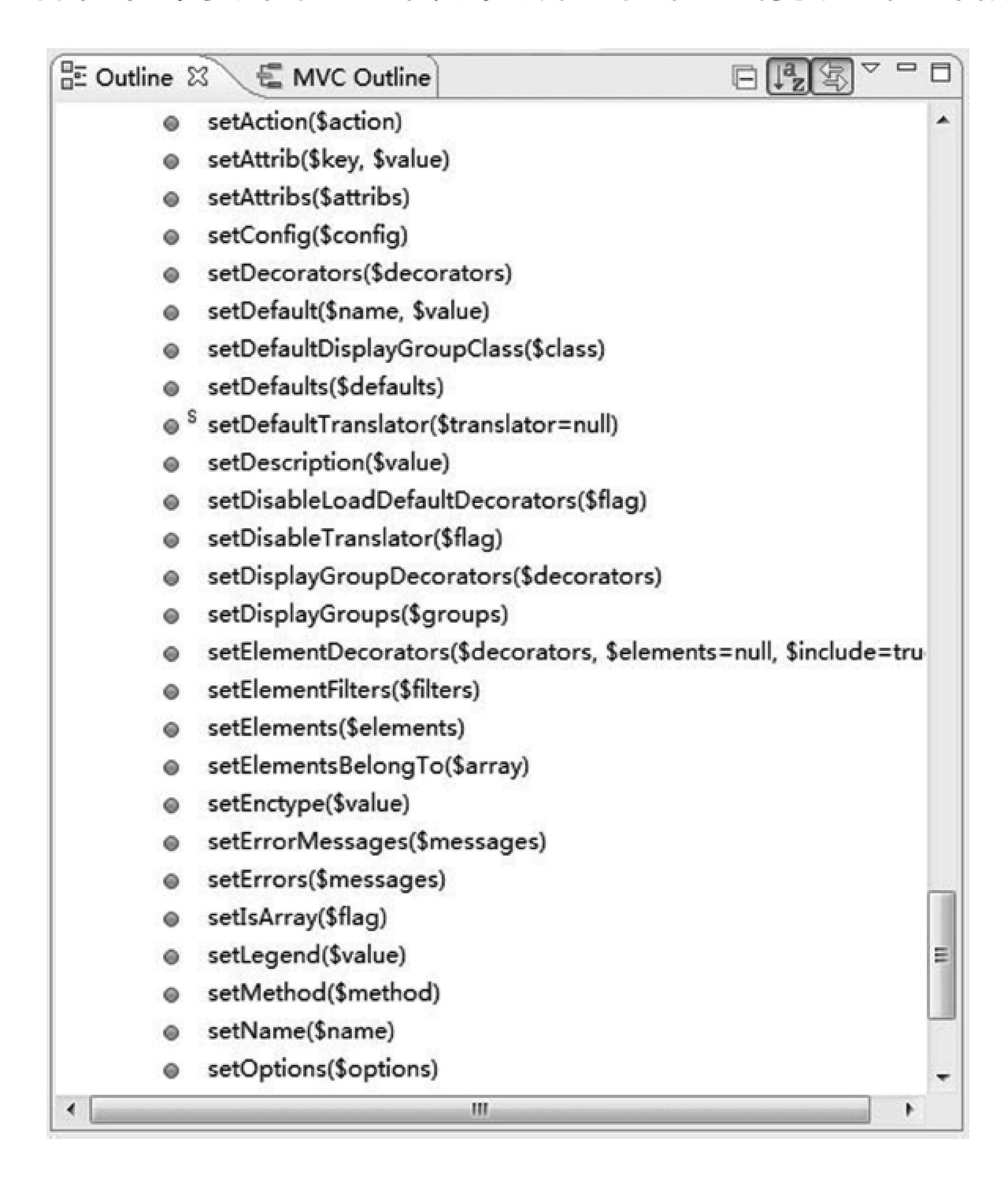


图 6.9　Zend_Form 类方法

6.2.3　Zend_Form 表单实例

为了讲解方便，上面的登录表单只用到了 text、password 和 submit 3 种类型的表单元素，下面通过用户注册表单的创建来学习使用更多类型的表单元素。

本书案例是一个办公自动化管理系统，这种应用一般是不设用户注册功能的，用户的注册由管理员在后台批量注入，在前台用户模块中只设置用户修改或完善个人信息的相关操作。这里创建一个用户信息表单，用作“用户登录”及“用户个人信息修改”功能实现的共用表单。

1. 用户信息表单的创建

用户信息根据应用项目的需求进行设置，主要有姓名、密码、性别、年龄、民族、邮箱、简介、工号、职称、照片、用户状态、用户角色等。为了简单，这里选用上述部分字段，创建步骤如下：

(1) 打开 Zend Studio 集成开发环境,选择 wmProject 项目。

(2) 选择 Project|Zend Tool 菜单项,打开 ZF Tool 工具命令窗口,输入命令:

```
Zf create form User
```

(3) 打开 application\forms\User.php 文件,添加代码:

```
class Wm_Form_User extends Zend_Form
{
    public function init()
    {
        //表单属性
        $this->setName('form_user')
         ->setMethod('post')
         ->addAttribs(array(
                'style'=>'margin:5px; padding:5px'
        ));
        //用户登录名
        $netName = $this->createElement('text','netname');
        $netName->setLabel('登录名: ');
        $netName->setRequired(TRUE);
        $netName->addValidator('stringLength',false,array(5,20));
        $netName->addErrorMessage('登录名要求英文 5-20 个字母或 2-6 个汉字');
        $this->addElement($netName);
        //职工姓名
        $userName = $this->createElement('text','username');
        $userName->setLabel('职工姓名: ');
        $userName->setRequired(TRUE);
        $userName->addValidator('stringLength',false,array(5,20));
        $userName->addErrorMessage('要求英文 5-20 个字母或 2-6 个汉字');
        $this->addElement($userName);
        //职工号
        $userId = $this->createElement('text','userId');
        $userId->setLabel('职工号: ');
        $userId->setRequired(TRUE);
        $userId->addValidator('stringLength',false,array(6));
        $userId->addErrorMessage('职工号要求至少 6 个字符');
        $this->addElement($userId);
        //密码
        $password = $this->createElement('password','password');
        $password->setLabel('密码: ');
        $password->setRequired(TRUE);
        $password->addValidator('stringLength',false,array(6));
        $password->addErrorMessage('密码要求至少 6 个字符');
        $this->addElement($password);
        //确认密码
        $password2 = $this->createElement('password','password2');
        $password2->setLabel('确认密码: ');
        $password2->setRequired(TRUE);
        $password2->addValidator('identical',false,array('token'=>'password'));
        //验证两个密码是否相同
        $password2->addErrorMessage('两次输入的密码密要相同');
        $this->addElement($password2);
        //性别
```

```
$ sex = $ this -> createElement('radio', 'sex');
$ sex -> setLabel('性别: ');
$ sex -> addMultiOptions(array(1 => '男', 0 => '女'));
$ sex -> setSeparator("");
$ this -> addElement( $ sex);
//年龄
$ userAge = $ this -> createElement('text', 'userage');
$ userAge -> setLabel('年龄: ');
$ userAge -> setRequired(TRUE);
$ userAge -> addValidator('NotEmpty', true);
$ userAge -> addErrorMessage('用户年龄不能为空');
$ this -> addElement( $ userAge);
//所属部门
$ department = $ this -> createElement('text', 'department');
$ department -> setLabel('所属部门: ');
$ department -> setRequired(TRUE);
$ department -> addValidator('stringLength', false, array(5,20));
$ department -> addErrorMessage('部门要求英文 5 - 20 个字母或 2 - 6 个汉字.');
$ this -> addElement( $ department);
//入职时间
$ registdate = $ this -> createElement('text', 'registdate');
$ registdate -> setLabel('入职时间: ');
$ registdate -> setRequired(TRUE);
$ registdate -> addValidator('NotEmpty', true);
$ registdate -> addErrorMessage('用户入职时间不能为空');
$ this -> addElement( $ registdate);
//电子邮件
$ email = $ this -> createElement('text', 'email');
$ email -> setLabel('电子邮箱: ');
$ email -> addValidator('EmailAddress');
$ email -> addErrorMessage('请输入一个有效的 email 地址');
$ this -> addElement( $ email);
//个人简介
$ profile = $ this -> createElement('textarea', 'profile');
$ profile -> setLabel('个人简介: ');
$ profile -> setAttribs(array('cols' => 40, 'rows' => 8));
$ this -> addElement( $ profile);
//照片
$ avatar = $ this -> createElement('image', 'avatar');
$ avatar -> setLabel('照片: ');
$ avatar -> setAttrib('src', '/images/pic01.jpg');
$ this -> addElement( $ avatar);
//用户状态
$ status = $ this -> createElement('select', 'status');
$ status -> setLabel('用户状态');
$ status -> addMultiOptions(array(
    '1' => '激活',
    '0' => '锁定',
));
$ this -> addElement( $ status);
//用户角色
$ role = $ this -> createElement('select', 'role');
$ role -> setLabel('选择角色: ');
$ role -> addMultiOptions(array(
```

```
                'user' => '普通用户',
                'editor' => '部门管理员',
                'subadmin' => '部门负责人',
                'admin' => '系统管理员'
            ));
            $role -> setRequired(TRUE);
            $this -> addElement($role);
            //提交按钮
            $submit = $this -> createElement('submit','submit');
            $submit -> setLabel('登录');
            $this -> addElement($submit);

    }
```

2. 用户信息表单的显示

用户信息表单作为登录、注册与用户信息修改的通用表单，在不同的使用环境中显示的内容是不同的。例如作为"用户登录"表单使用时，仅使用"登录名"、"密码"、"确认密码"、"提交"4个表单元素；用作"用户个人信息修改"表单时，不能显示"用户名"等一些用户没有权限修改的表单元素；而在用作系统管理员的"用户注册信息"修改表单时，需要显示全部的表单元素。也就是说，不同的用户权限、不同的使用场合，需要显示不同的内容。

1）全部显示

按照前面介绍的方法分别在控制器、视图中添加代码。

(1) 用ZF Tool工具在User控制器中创建register方法，并添加如下代码：

```
class UserController extends Zend_Controller_Action
{
    …
    public function registerAction()
    {
        $formUser = new wm_Form_User();
        $this -> view -> formUser = $formUser;
    }
}
```

(2) 修改视图代码。打开User控制器的register方法对应的视图文件application\views\scripts\user\register.phtml，删除原有代码，添加新代码：

```
<br /><br />
<?php echo $this -> formUser; ?>
```

(3) 启动Apache服务器，在浏览器的地址栏中输入http://wmoams.com/user/register，页面效果如图6.10所示。

2）登录显示

打开User控制器的login方法，修改原有代码，使用新建的user表单替换原有的login表单，并移除user表单中不必要的表单元素。代码如下：

```
public function loginAction()
    {
        //关闭默认布局模板
        $this -> _helper -> layout -> disableLayout();
```

```
        //使用 User 表单
        $formLogin = new wm_Form_User();
        //移除不必要的表单元素
        $formLogin->removeElement('username');
        $formLogin->removeElement('userId');
        $formLogin->removeElement('userage');
        $formLogin->removeElement('sex');
        $formLogin->removeElement('department');
        $formLogin->removeElement('registdate');
        $formLogin->removeElement('email');
        $formLogin->removeElement('avatar');
        $formLogin->removeElement('status');
        $formLogin->removeElement('profile');
        $formLogin->removeElement('role');

         if($this->getRequest()->isPost()){
            if($formLogin->isValid($_POST)){
                $data = $formLogin->getValues();
                if($data['username'] == "wmstudio" &&
                              $data['password'] == "123456")
                    $this->redirect('/index/main');
                else
                    $this->view->message = "用户名或密码错误!";
            }
        }
        $this->view->formLogin = $formLogin;
    }
```

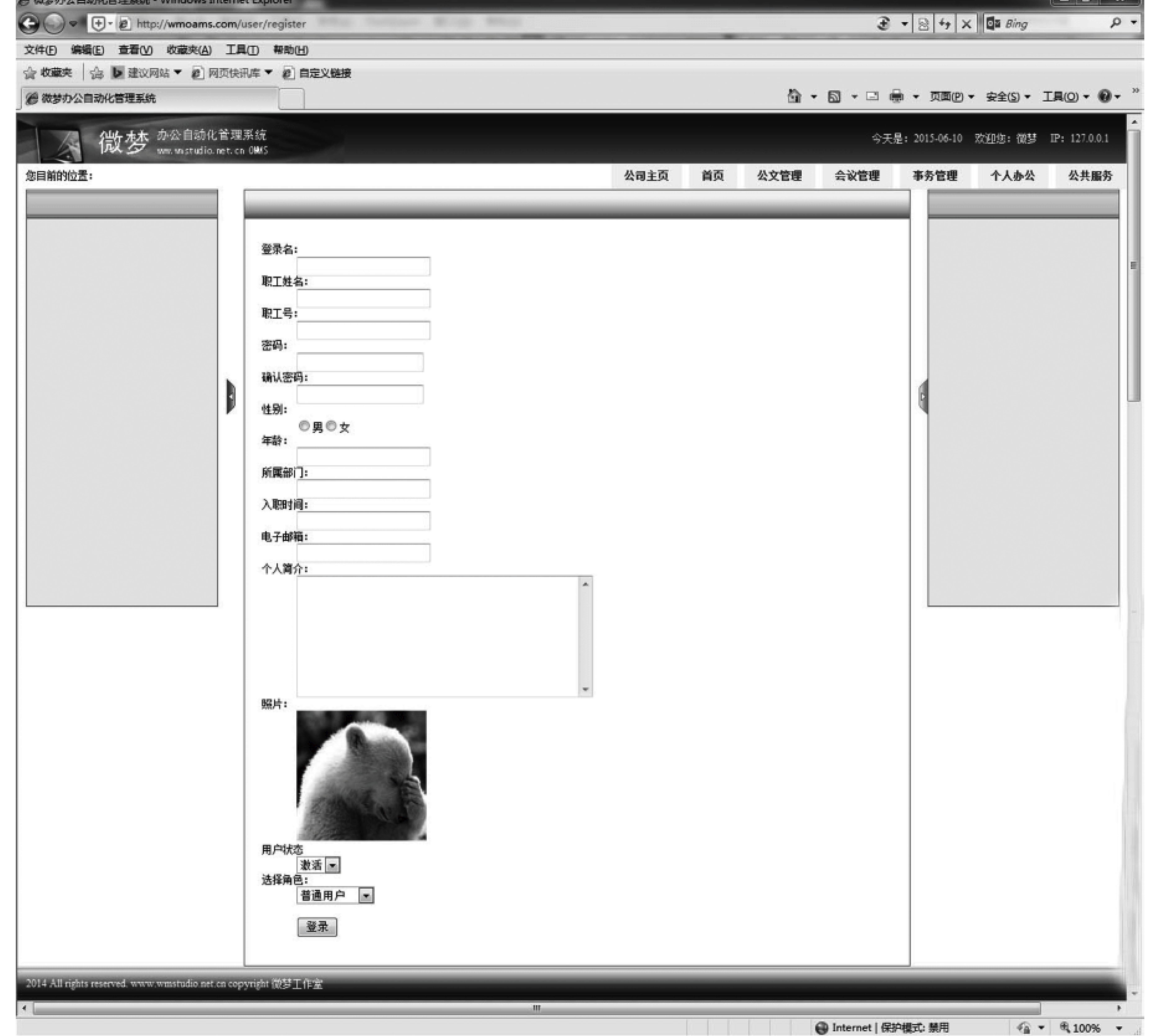


图 6.10　用户信息表单页面

使用 user 表单的登录页面效果如图 6.11 所示。



图 6.11　使用用户信息表单的登录页面

从上面的案例可以看出,使用 Zend Form 表单不仅能够增强代码的重复利用,提高项目的开发效率,而且通过它添加、删减系统功能也非常方便,这也显示出了采用面向对象的编程方式以及 MVC 的开发模式进行项目开发的突出优势。

6.3　Zend_Form 表单装饰器

在前面 6.1.2 节登录表单的显示中采用了重新定义<dl>、<dd>、<dt>标签样式的方式对界面进行了简单的布局,效果显示这种方法是可行的。Zend Framework 提供了另一种比较好的表单布局方法,通过添加表单装饰器来控制表单的显示布局。

6.3.1　Zend_Form 表单装饰器的类型

Zend_Form 表单使用以下 5 个装饰器。

(1) ViewHelper:使用一个标准表单视图帮助器来呈现表单元素,并指定一个视图助手用于解析元素。在默认情况下,Zend_Form_Element 指定 formText 视图助手。

(2) Errors:通过一个无序列表呈现验证错误,使用 Zend_View_Helper_FormErrors 追加错误消息给表单元素,如果没有错误,就不追加。

(3) Description:呈现附加在元素上的相关描述,通常用于提示 tooltips。

(4) HtmlTag:默认用一个<dd>标签包括以上(1)、(2)、(3)中的全部内容。

(5) Label:呈现放置在以上内容之前的标签 label,默认用一个<dt>标签包围。即使用 Zend_View_Helper_FormLabel 预先准备一个标签给元素,并把它封装在一个<dt>标签里。

在上面的 5 个装饰器中,Description 用于信息提示,用得非常少,常用的是另外 4 个。这些装饰器对存储在表单元素里的元数据的片段进行操作。

6.3.2　Zend_Form 表单装饰器的工作原理

一般情况下,在表单元素对象的初始化过程中加载默认的表单元素装饰器,其加载顺序依据各个装饰器的注册顺序,所以在注册 Zend_Form 表单装饰器时一定要注意它们的添加

顺序。当然，也可以通过传递 disableLoadDefaultDecorators 属性给表单元素对象的构造方法来关闭它，例如：

```
$ element = new Zend_Form_Element('foo',
array('disableLoadDefaultDecorators' => true));
```

对于 Zend_Form 表单，初始内容由 ViewHelper 装饰器生成，它生成表单元素自己；接着 Errors 装饰器从元素里抓取错误消息，如果有任何错误，就传递给 FormErrors 视图助手来解析；下一个装饰器 HtmlTag 在一个 HTML <dd>标签里封装元素和错误；最后，Label 装饰器读取元素的标签并传递给 FormLabel 视图助手，封装在一个 HTML <dt>标签里。即 Zend_Form 表单装饰器的工作分成以下 4 个步骤：

(1) ViewHelper 分离出每个元素并呈现它；

(2) Errors 分离验证错误并用一个无序列表呈现它们；

(3) HtmlTag 生成一个 HTML 标签，把以上两者包起来(默认使用<dd>标签)；

(4) Label 装饰器分离 label 标签并呈现它(默认使用<dt>标签)。

这就类似于一个洋葱，从里到外按顺序层层包裹，所以我们可以形象地把表单装饰的过程看作从内到外创建洋葱的过程，如图 6.12 所示。

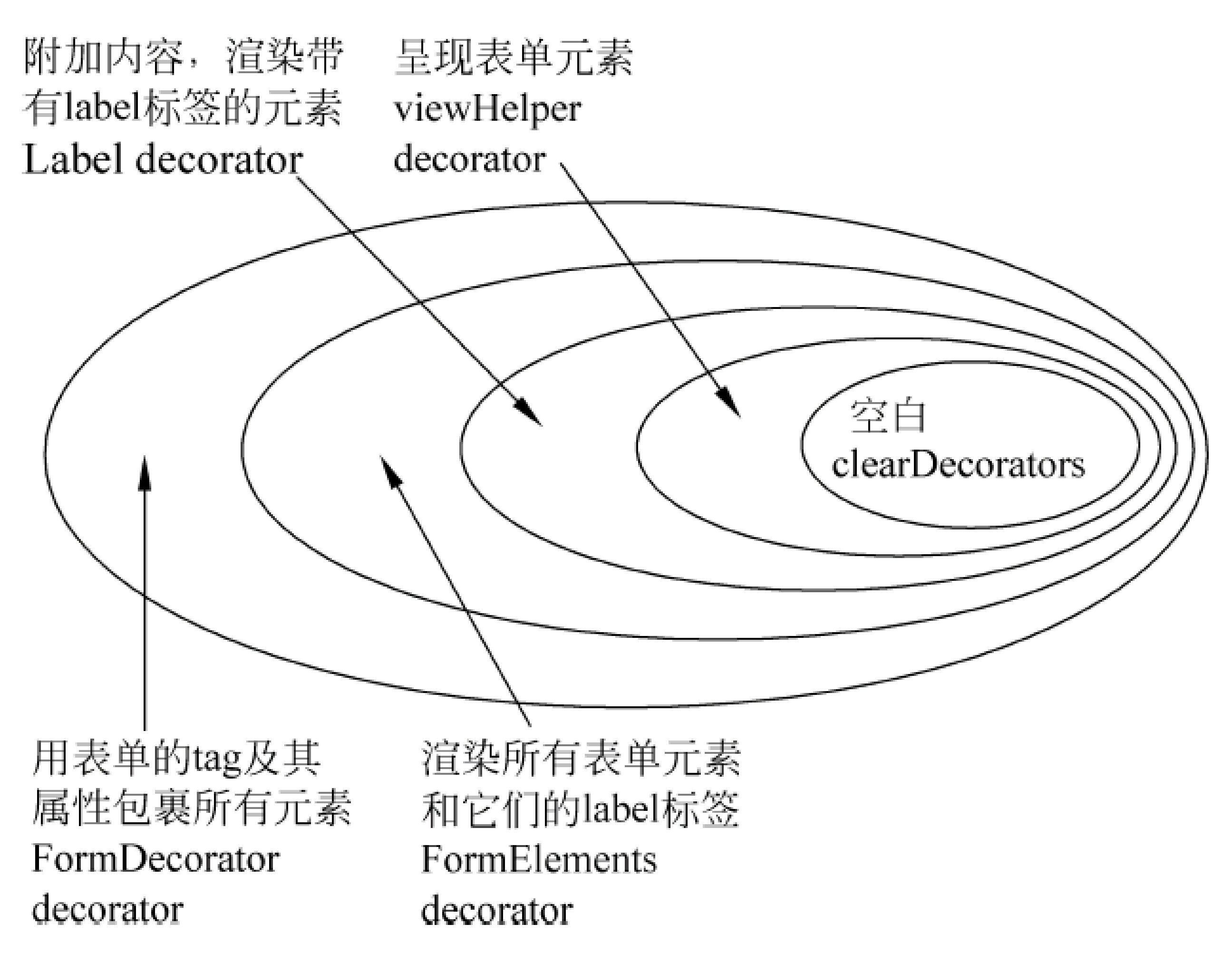


图 6.12　表单装饰器原理

6.3.3　Zend_Form 表单装饰器实例

下面用 Zend_Form 表单装饰器来布局 6.2 节中创建的 user 表单。

打开 application\forms\User.php 表单文件，在 Wm_Form_User 表单类的 init 初始化方法末尾添加如下代码：

```
public function init()
{
        …
        //提交按钮
        $ submit = $ this -> createElement('submit','submit',array('label' =>'提交','ignore' =>
true));
```

```
        $this->addElement($submit);
        //表单装饰器
        $this->setElementDecorators(array(
                'ViewHelper',
                'Errors',
                array(array('data'=>'HtmlTag'),array('tag'=>'td')),
                array('label',array('tag'=>'td')),
                array(array('row'=>'HtmlTag'),array('tag'=>'tr'))
        ));
        $this->setDecorators(array(
                'FormElements',array('HtmlTag',array('tag'=>'table','class'=>'usertable'),'Form')
        ));
    }
```

以上代码依照上面介绍的从里到外的次序重新设置了表单装饰器，先用 Zend_Form 类中的 setElementDecorators()方法改写表单元素装饰器，ViewHelper 和 Errors 采用默认设置，把默认的 HtmlTag 和 Lable 改成了<td>单元格标签，然后多加一行代码，在以上的全部代码外面再包上一层，用<tr>(也就是表格中的"行")作为标签把它们包裹了起来。

接着用 Zend_Form 类中的 setDecorators()方法对表单进行设置，在表单的最外层套上了<table>标签，同时还给它设定了 CSS 的 class 类样式，这样表单就被放在表格中了，从外到里，首先是<table>，再往里是<tr>行，每行里又有两个<td>，分别放着 Label 和表单元素，这就是大家非常熟悉的表格布局方式。添加了装饰器之后的 user 表单页面效果如图 6.13 所示。

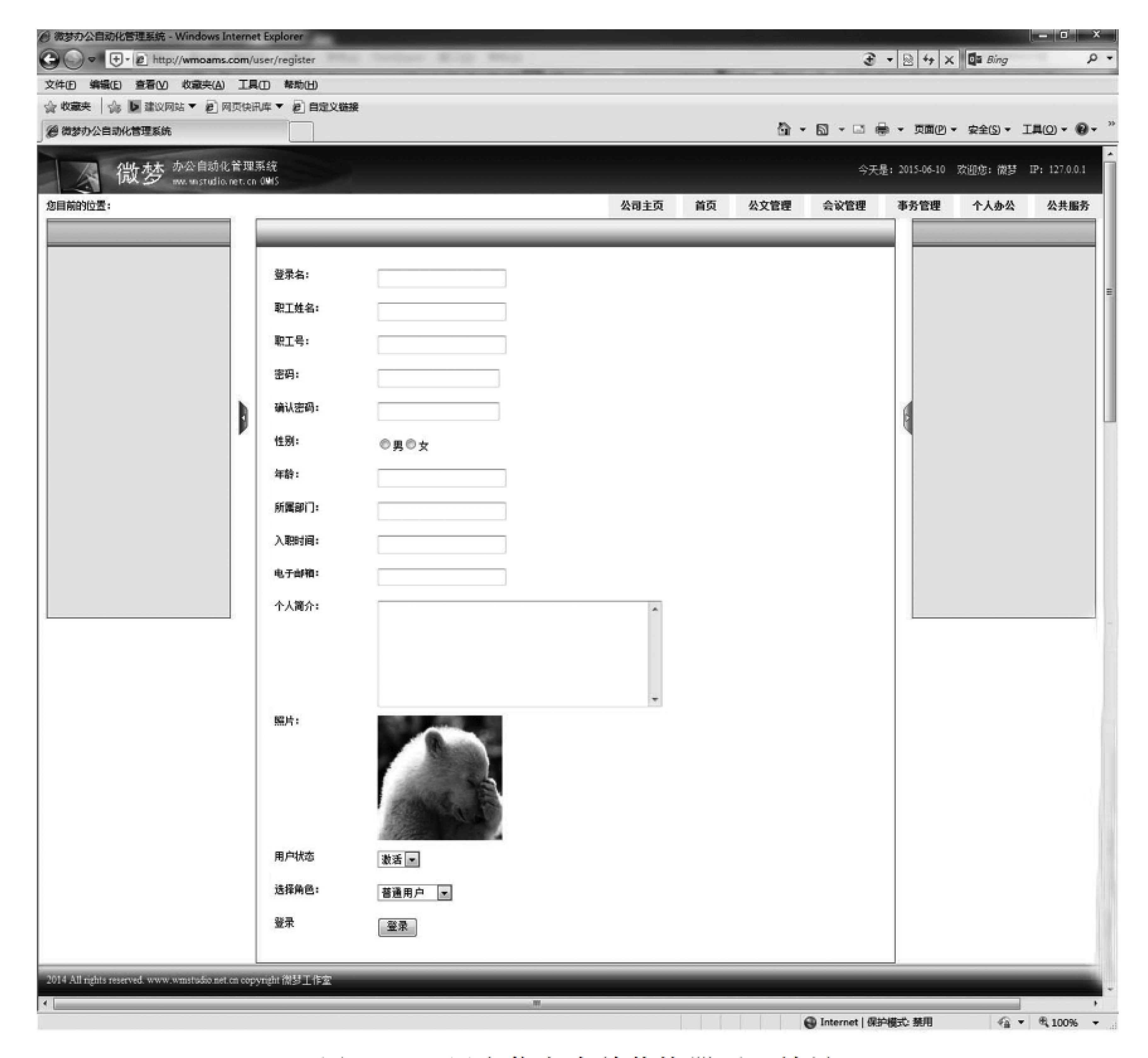


图 6.13　用户信息表单装饰器页面效果

6.4 Zend_Auth 认证

在 6.2 节中对登录表单的处理只是一个简单的验证，即对用户输入的数据的有效性进行了检验，并没有对用户身份进行认证。

用户登录其实就是一个寻求认证的过程。用户在登录表单中输入一些必要的数据，程序从数据库中查询这些数据是否满足认证条件，如果满足则允许登录，并将一些用户信息注册到会话或其他存储对象中；如果不满足，则拒绝登录。Zend Framework 提供了 Zend_Auth 组件实现用户登录认证。

6.4.1 Zend_Auth 适配器

1. Zend_Auth 简介

Zend_Auth 组件是 Zend Framework 提供的用户访问认证组件，称为访问认证适配器。该适配器依靠特定的认证服务(例如 LDAP、RDBMS 或基于文件的存储)对内容进行认证。基于不同认证服务的适配器会有不同的选项和行为，但有些基本的内容在不同的认证适配器中是相同的。例如，接受认证证书、依靠认证服务执行查询以及返回结果等。

每个 Zend_Auth 适配器类都实现了 Zend_Auth_Adapter_Interface 认证基本接口。该接口定义了 authenticate()方法，适配器都必须实现该方法。在调用适配器的 authenticate 方法之前，每个适配器还必须进行初始化，包括设置证书(如用户名和密码)、为适配器专用的配置选项定义一些值(如为数据库工作表适配器做的连接设置)等。

2. 常用认证适配器

常用的系统认证适配器有 3 种，即摘要式认证 Zend_Auth_Adapter_Digest、数据库认证 Zend_Auth_Adapter_DbTable 以及 HTTP 认证 Zend_Auth_Adapter_Http。

摘要式认证是一种 HTTP 认证的方法，该方法通过不需要使网络传递明文密码的方法对基本认证加以改进。Zend_Auth_Adapter_Digest 适配器允许依靠文本文件进行认证。该文本文件包括数行摘要式认证的基本元素，每一行的内容形如 user：realm：ebbcoff9a121dbb6789bbe5f82174fa0。其中，user 为用户名，realm 为领域，最后为密码的 MD5 值。

数据库认证是最重要的认证方法，Zend_Auth 中的适配器 DbTable 用于提供依靠数据库表中的记录内容进行认证的功能。该适配器需要 Zend_Db_Adapter_Abstract 的实例(数据库适配器)传递给它的构造方法，所以每个实例要和特定的数据库连接绑定。

Zend Framework 的 Zend_Auth_Adapter_Http 类支持符合 RFC-2617 规范的基本认证和数字 HTTP 认证。数字认证是在基本认证的基础上改进而来的，是不需要在网络上传输明文密码的一种认证方式。如果要进行 HTTP 认证，还需要为 Zend_Auth_Adapter_Http 对象提供 Resolver。Resolver 的作用是接收用户名和领域，并返回证书值。基本认证通常期望接收用户密码的 Base64 编码版本。数字认证期望接收用户的用户名、领域和密码的 hash 值。

6.4.2 Zend_Auth 认证的实现

在 Zend Framework 提供的 3 种常用认证方法中，我们采用数据库认证的方式来实现系统的用户登录认证。

1. 创建用户信息数据表

根据第 3 章中给出的总体设计方案在数据库中创建一个用户信息表。

打开 phpMyAdmin 数据库管理工具，在 db_wmoams 数据库中新建名为 tb_users 的数据表，其字段含义见表 6.2，字段属性如图 6.14 所示。

表 6.2 tb_users 数据表字段说明

字 段 名	说 明
id	自增变量，主键，用户在系统中的唯一标识
netame	用户登录名
staffNo	职工工号
username	职工姓名
password	登录密码
sex	职工性别
age	职工年龄
nation	民族
birth	出生年月
idcard	身份证号码
education	学历
degree	学位
department	所属部门
depart_id	部门编号
post	职务
entrytime	入职时间
phone	电话
email	邮箱
profile	个人简介
avatar	照片
role	用户角色
status	用户状态
time_reg	用户注册时间
time_last	用户最后登录时间

创建用户信息数据表后，在表中插入一些数据，以便于后续的测试。例如登录名 wmstudio，密码 e10adc3949ba59abbe56e057f20f883e（即 123456）。注意密码为 MD5 加密值，也可以用其他加密算法。

2. 实现数据库认证

在 Zend Studio 集成开发环境中打开 application\controllers 目录中的 UserController.php 文件，先在 User 控制器的 login 方法中删除如下代码：

```
if( $ formData[ 'netname'] == 'wmstudio' &&
                                    $ formData[ 'password'] == '123456')
        $ this -> redirect( '/index/main');
else
        $ message . = "用户名或密码错误!";
```

然后在该位置添加新代码：

```
//取得默认的数据库适配器
$ db = Zend_Db_Table::getDefaultAdapter();
//实例化一个 Zend_Auth 适配器
$ authAdapter = new Zend_Auth_Adapter_DbTable(
                              $ db, 'tb_users', 'username', 'password');
//设置认证登录名及密码
$ authAdapter -> setIdentity( $ formData[ 'netname']);
$ authAdapter -> setCredential(md5( $ formData[ 'password']));
//执行认证
$ authResult = $ authAdapter -> authenticate();
if( $ authResult -> isValid()){
        $ this -> redirect( '/index/main');
}else{
        $ message . = "用户名或密码错误!";
}
```

名字	类型	排序规则	属性	空	默认	额外
id	int(10)		UNSIGNED	否	无	AUTO_INCREMENT
netname	varchar(20)	utf8_general_ci		是	NULL	
staffNo	char(6)	utf8_general_ci		否	无	
username	varchar(20)	utf8_general_ci		否	无	
password	char(32)	utf8_general_ci		是	NULL	
sex	tinyint(1)			是	NULL	
age	int(4)			是	NULL	
nation	varchar(20)	utf8_general_ci		是	NULL	
birth	varchar(100)	utf8_general_ci		是	NULL	
idcard	bigint(18)			是	NULL	
education	char(10)	utf8_general_ci		是	NULL	
degree	char(10)	utf8_general_ci		是	NULL	
department	varchar(100)	utf8_general_ci		是	NULL	
depart_id	int(10)		UNSIGNED	是	NULL	
post	char(20)	utf8_general_ci		是	NULL	
entrytime	datetime			是	NULL	
phone	varchar(20)	utf8_general_ci		是	NULL	
email	varchar(255)	utf8_general_ci		是	NULL	
profile	text	utf8_general_ci		是	NULL	
avatar	varchar(255)	utf8_general_ci		是	NULL	
role	varchar(20)	utf8_general_ci		是	NULL	
status	tinyint(1)			是	NULL	
time_reg	int(11)			是	NULL	
time_last	int(11)			是	NULL	

图 6.14　用户信息表属性设置

从上述代码可以看出，Zend_Auth 认证分为下面 5 个步骤。

1）获取数据库适配器

在 6.4.1 节中讲过，采用数据库认证方法时必须绑定数据库连接。这里通过 Zend_Db_Table 类的静态方法 getDefaultAdapter 获取配置文件中的默认数据库适配器对象。

2）实例化认证适配器

实例化一个 Zend_Auth_Adapter_DbTable 适配器对象 authAdapter，在实例化时通过配置该适配器的属性进行初始设置。从代码中可以看出此适配器有 4 个属性，分别是数据库对象（zendDb）、数据表名（tableName）、身份列（identlbColunm）和证书列（credentialColunm）。

- zendDb：指定数据库适配器。
- tableName：包含认证证书的数据库表名，执行数据库认证查询需要依靠这个证书。
- identityColumn：数据库表的字段名称，用来表示身份。身份列必须包含唯一的值，例如用户名或者 email 地址等。
- credentialColunm：数据库表的字段名称，用来表示证书。在一个简单的身份和密码认证方式下，证书的值对应用户密码。

这里的代码给出的上述 4 个参数分别是 db、tb_users、netname 和 password。

3）设置认证身份与证书

使用认证适配器 authAdapter 对象的 setIdentity 方法将表单提交的“用户登录名”数据 netname 设置为身份；使用其 setCredential 方法将密码设为证书，并用 MD5 算法加密。

4）执行认证

以上各项参数设置完成后，通过认证适配器 authAdapter 对象的 authenticate 方法执行认证，并得到返回的认证结果。

5）判断认证结果，确定路由路径

获得认证结果以后，通过认证结果对象 Zend_Auth_Auth_Result 的 isValid 方法判断认证是否成功，然后确定页面的跳转方向。

在调试过程中，如果认证不成功，可以通过 Zend_Auth_Auth_Result 类的 getCode、getMessages 方法获取认证结果内容，判断认证失败的原因。

6.5 登录功能的完善

通过上面几节的介绍，系统登录功能已经基本完成，但还存在一些不太完美的细节。例如，当登录信息验证没通过时，给出的错误提示信息与表单元素纠缠在一起，破坏了表单的整体布局；用户身份认证通过后，还需要对用户的某些信息进行保存，以方便后续页面的认证等。下面来完善这些细节。

6.5.1 验证信息的集中显示

如图 6.2、图 6.3 所展示的界面那样，将登录错误信息在一个信息提示框中集中显示。

1. 修改视图文件

在 Zend Studio 集成开发环境中打开视图文件 application\views\scripts\user\login.phtml,添加如下代码:

```
<div id="login_center">
    <div class="left"><img src="/images/login_center_mark.gif" /></div>
        <div class="center"></div>
        <div class="right"><?php echo $this->formLogin; ?></div>
        <?php if(trim($this->message) != ''):?>
        <div style="position:absolute;right:10px;width:250px;
                                    height:250px;background:#ff0">
            <div style="width:80%; height:75%;background:#fff;
                 margin:20px auto;padding:10px;border:1px solid #cc0">
                 <?php echo $this->message; ?>
            </div>
        </div>
<?php endif;?>
</div>
```

上述代码中的变量 message 是从 User 控制器的 login 方法中传递过来的用户登录表单验证信息。

2. 修改控制器代码

在 Zend Studio 集成开发环境中打开 application\controllers\User.php 控制器文件,修改或添加 User 控制器的 login 方法中的代码,例如:

```
$message = '';
if($this->getRequest()->isPost()){
   $formData = $this->getRequest()->getParams();
   if($formLogin->isValid($formData)){
        …
   }else{
        $formLogin->populate($formData);
        $arrayMessage = $formLogin->getMessages();
        if (isset($arrayMessage['netname']['0'])) {
          $message .= '   '.$arrayMessage['netname']['0'].'<br />';
        }
        if (isset($arrayMessage['password']['0'])) {
          $message .= '   '.$arrayMessage['password']['0'].'<br />';
        }
        if (isset($arrayMessage['password2']['0'])) {
          $message .= '   '.$arrayMessage['password2']['0'].'<br />';
        }
   }
}
$this->view->formLogin = $formLogin;
$this->view->message = $message;
```

6.5.2 认证信息的保存

在第 4 章的 4.1.5 节中已经初始化了一个认证对象,这里只需要将认证信息存入该对象即可。

在 Zend Studio 集成开发环境中打开 User 控制器的 login 方法，添加如下代码：

```
…
if( $ authResult -> isValid()){
      //获取 Zend_Auth 认证对象
       $ auth = Zend_Registry::get('auth');
       //保存认证结果信息
       $ auth -> getStorage() -> write(
                        $ authAdapter -> getResultRowObject(
                              array('id','username','department','role')
                  ));
      $ this -> redirect('/index/main');
}else{
      $ message . = "用户名或密码错误!";
}
```

在第 4 章的 4.1.5 节中设置认证对象时，将该对象以 auth 的名字存储在了注册表对象中。这里可以通过 Zend_Registry 类的静态方法 get 直接取出，然后通过 Zend_Auth_Storage_Session 类的 write 方法将认证的用户信息保存。这样，在系统的其他地方就可以随时获取到登录用户信息了。

6.5.3 认证信息的使用

上面详细介绍了用户登录及认证过程，并得到了认证信息，但这些信息要在页面中使用才能发挥它的拦截作用。例如，在浏览器中直接输入 http://wmoams.com/index/main，同样能访问到系统主页，请求绕过了登录页面，很明显这是不允许的。

本系统为办公系统，用户必须通过认证后才能进入系统页面，所以，系统中的每一个页面都必须验证用户是否登录。通过前面的学习，我们已经非常清楚 Zend Framework 项目的路由规则，访问系统中的任何页面都必须通过相应的控制器，因此，可以在每个控制器中完成用户是否登录的判断。下面解决绕过登录页面直接访问系统主页的问题。

在 Zend Studio 集成开发环境中打开 Index 控制器文件 application\controllers 下的 IndexController.php，添加如下代码：

```
class IndexController extends Zend_Controller_Action
{
   protected $ _user = null;
    public function init()
    {
         $ auth = Zend_Registry::get('auth');
          if( $ auth -> hasIdentity()){
              $ this ->_user = $ auth;
         }
    }
…
    public function mainAction()
    {
```

```
        if( $ this -> _user){
              $ this -> render();
         }else {
              $ this -> redirect('/');
              exit();
         }
    }
}
```

为了方便，先定义了一个控制器变量_user，并在 init 方法中用存储在注册表对象中的认证对象对它进行初始化。接着，在 main 方法中通过_user 变量的值进行页面跳转，若用户经过了认证，继续访问 main 方法对应的视图页面；若用户没有经过认证，则跳转到系统首页，这里也可以让页面跳转到登录页面，即在 redirect 方法中设置访问地址/user/login。

通过上述修改后，在浏览器中输入 http://wmoams.com/index/main，页面跳转至系统封面首页，即有效地制止了用户的非法访问。

6.5.4 账户注销

用户登录后还需要能够随时注销，实现用户注销非常简单，只需在 User 控制器添加一个方法，销毁登录时创建的认证信息就可以了。

启动 Zend Studio 集成开发环境，选择项目工作区的 wmProject 项目中的某个文件或文件夹，打开 Zend Tool 工具命令窗口，输入命令：

```
zf create action logout User
```

在 User 控制器中创建 logout 方法，并添加代码：

```
public function logoutAction()
{
     $ auth = Zend_Registry::get('auth');
     $ auth -> clearIdentity();
     $ this -> redirect('/');
 }
```

上述阴影代码中的第 1 句获取 Zend_Auth 认证对象，第 2 句清除该对象中的认证信息，第 3 句使页面跳转到系统首页。

账户注销应该在系统中的任何页面都能进行，页面中的“注销”链接应放在系统的布局模板中。打开 application\layouts\header.phtml 文件，添加代码：

```
<div id = "header">
     <div id = "logo">
          <img src = "/images/logo.jpg">
          <div id = "usepanel">
               <a href = '/user/logout'>注销</a>
          </div>
     </div>
</div>
```

6.6 本章小结

本章通过系统登录系列功能的实现详细介绍了 Zend Framework 的表单组件 Zend_Form 以及认证组件 Zend_Auth，主要内容包括表单的创建、验证器及样式的设置、表单的装饰、表单的显示与数据处理，以及用户登录认证的数据库认证方法。

表单操作是 Web 应用开发中最重要也是最常用的功能，是本章的学习重点。Zend Framework 的 Zend_Form 组件利用面向对象的封装技术将以往凌乱的 HTML 代码和与之相应的验证、处理等复杂操作提升到了一个新的层次，使之变得十分简单，也更容易理解。

第7章 用户管理及 Zend Framework 模块

用户作为应用系统的使用者与维护者，是系统的主要服务对象，也是系统生存与发展的重要因素。办公自动化管理系统中的用户信息管理一般分为前台管理与后台管理两个部分。前台管理主要是用户对自身信息的维护，而后台管理，则是系统管理员对系统所有用户的信息的管理，包括用户的注册、删除、更改以及各种各样的查询等。

本章通过实现案例项目的用户信息管理功能，介绍 Zend Framework 的模块以及职工信息的增、删、改、查等数据库操作。

7.1 系统后台管理模块

在第2章中介绍 Zend Framework 项目结构的时候，我们曾经提到在 Zend Framework 项目的目录结构中可能会有一个名为 modules 的文件夹(见表 2.1)，这里存放的文件，即系统的模块文件。

7.1.1 Zend Framework 模块概述

应用系统中的“模块”一般指“功能模块”。对于 Zend Framework 来说，一个模块一般是一个大的功能区，几个模块按照一定的规则相互联系构成更加复杂的系统或过程。在开发过程中，通过制定规则和约定将系统划分成模块安排分工，可以极大地提高开发效率。

模块是一个相对的概念，对其划分没有一个固定的规则，要结合应用系统的规模、精细化程度、开发人数和分工、开发路线图等因素来考虑。需要注意的是，模块并不是划分得越细越好，应该以满足需求、方便开发人员相互协作等作为划分标准。模块划分得太细，不仅会产生运行效率问题，而且会给开发工作带来困扰；如果划分得太粗，又会导致应用程序功能局部臃肿，耦合度增高，使后期维护成本增加，甚至直接影响应用程序的生命周期。

应用系统中的不同模块之间应保持尽量低的耦合度，在需要的时候，各个模块之间应尽可能地相对独立，这样，一个模块出现问题，不会大面积影响其他模块。这里讲的模块“独立”是针对代码的组织层面而言的，在内在逻辑层，各个模块是不能完全独立的，所以对模块低耦合度的追求也要与应用程序的特点和规模、代码的重用等结合起来考虑，不宜过度。

7.1.2 模块的创建

应用系统一般分为前台与后台，前面几章我们所做的开发均属于前台模块。这里使用了“模块”这个词，也就是说前面编写的所有程序实际上也是一个模块，它就是默认模块 default。大家可以在浏览器的地址栏中输入 http://wmoams.com/default/user/login 来

访问系统的登录页面，请求是有效的。这说明 default 模块是存在的，只是因为它是默认模块，就像默认控制器 Index、默认方法 index 一样，可以在访问路径中予以省略。

下面使用 ZF Tool 工具创建一个后台管理模块。启动 Zend Studio 集成开发环境，定位到案例项目 wmProject 中，打开 ZF Tool 工具，在命令窗口中输入：

```
zf create module admin
```

即在项目中创建了一个名为 admin 的模块，注意模块名大小写敏感。

命令执行完毕后，展开项目的 application 目录，可以看到在该目录下新创建了一个名为 modules 的目录，里面又有一个 admin 文件夹，其中包括 controllers、models、views 3 个空目录，将分别用来存放该模块的控制器、模型和视图文件。很明显，新模块 admin 也将采用 MVC 的模式来组织。

另外，ZF Tool 工具还在 application.ini 配置文件中添加了一行配置代码来指定模块的访问路径：

```
resources.frontController.moduleDirectory = APPLICATION_PATH "/modules"
```

在前面的程序开发过程中，我们使用的名字空间为 Wm_，它是作用于默认模块 default 上的。在新的模块中，应用程序将启用模块名作为名字空间。也就是说，在 Admin 模块中创建控制器、添加模型时都将以 Admin_作为前缀。为了能让应用自动搜索到这些资源，需要在 application.ini 文件中增加如下配置：

```
resouces.modules[] = ""
```

这样确保了模块作为资源被自动注册到配置文件中，模块也就能正常工作了。

7.1.3 控制器的创建与初始化

1. 创建控制器

在模块中创建资源的方法与前面相同，只是在命令中要加入模块名，用来指明该资源属于哪个模块。

打开 Zend Studio 集成开发环境中的 Zend Tool 工具命令窗口，分别输入命令：

```
Zf create controller Index 1 admin
```

和

```
Zf create controller User 1 admin
```

在模块 admin 中创建 Index、User 控制器，命令行中的“1”表示在创建控制器的同时添加 index 方法及视图。注意命令中必须加入 admin 模块名，否则控制器创建到默认模块（也就是 default 模块）中，就像我们以前做的一样。

打开新创建的控制器代码，它的类名与默认模块中的控制器类有所不同。在 admin 模块中创建的控制器类名均以 Admin_作为前缀，即将模块名首字母大写后作为前缀。例如：

```
class Admin_IndexController extends Zend_Controller_Action
```

```
{
    public function init()
    {
        /* Initialize action controller here */
    }
    public function indexAction()
    {
        //action body
    }
}
```

2. 初始化控制器

在 User 控制器中添加一些变量，并在初始化方法中给其赋值。这些变量是在以后的方法中需要用到的共性参数，在这里统一进行初始化，例如：

```
class Admin_UserController extends Zend_Controller_Action
{
    //登录用户
    private $_user = null;
    //是否登录标志
    private $_login = null;
    //职工信息
    private $_employee = null;
    //注册表对象
    private $_config = null;
    public function init()
    {
        $this->_helper->layout()->setLayout('admin');
        //初始用户表模型
        $this->_employee = new Wm_Model_User();
        $this->_config = Zend_Registry::get('config');
        $auth = Zend_Registry::get('auth');
        if($auth->hasIdentity()){
            $this->_user = $auth;
            $this->_login = true;
        }
    }
```

7.1.4 后台管理模板设计

系统的后台管理模块主要用于系统对象的管理与系统运行的维护。由于使用功能上的差异，它的页面布局与前面我们设计的前台模板应该是不一样的，所以有必要为新模块启用新的布局模板。

根据第 3 章中设计的系统总体结构，我们启用如图 7.1 所示的页面布局模式。

1. 模板页面设计

在目录 application\layouts\scripts 下新建 admin.phtml 模板文件，添加如下代码：

```
<?php echo $this->docType();?>
<html>
```

```
<head>
…
<?php echo $this->headTitle();?>
</head>
<body>
<div id="wrap">
  <?php echo $this->partial('adminHeader.phtml');?>
<div id="breadnav" style="width:500px;"><span>您目前的位置:
                              <?php echo $this->position?></span></div>
<div id="nav"></div>
<div id="main" style="width:100%;background:#ffffff">
    <div id="sidebar">
            <?php echo $this->partial('adminMenu.phtml');?></div>
    <div id="content" style="width:78%">
        <?php echo $this->layout()->content;?>
    </div>
    <div class="clear"></div>
</div>
<?php echo $this->partial('adminFooter.phtml');?>
</div>
</body>
</html>
```



图 7.1 用户信息页面模板效果图

该模板文件是在前台模板 layout.phtml 基础上修改而来的，代码中省略的部分与原模板相同。

2. 启用新模板

在 admin 模块的 Index、User 控制器的 init 初始化方法中添加如下代码，启用新的 admin.phtml 模板。

```
$this->_helper->layout()->setLayout('admin');
```

启动 Apache 服务器，在浏览器的地址栏中输入 http://wmoams.com/admin，即可看到新模板的页面效果。

3. 验证用户身份

用户必须登录后才能进入系统。由于采用的是统一登录界面，所以如果是具有管理员身份的用户登录，还需要在系统前台主页中添加链接，让用户能够进入系统后台管理中心。另外，在后台管理模块的每一个页面中还要添加判断用户是否登录的代码，如果用户不是从系统首页进入，则强制返回系统首页。

1）添加链接

打开 layout 模板文件 layou.phtml 中的 head.phtml 视图文件，在系统 logo 标识所在的 div 标签中添加“进入管理中心”链接，链接地址设置为/admin。

2）判断用户是否登录

在 admin 模块的 Index 和 User 控制器中分别添加代码：

```
public function indexAction()
    {
        if(!$this->_login){
            $this->redirect('/');
            exit();
        }
    }
}
```

以上是 admin 模块的 Index 控制器中的代码，User 控制器的代码与其相同，由于代码比较简单，这里就不多解释了。系统的访问控制，即用户的权限管理，将在第 12 章详细介绍。

7.2 用户信息的后台管理

用户信息的后台管理包括用户信息的查询、添加、更改、删除以及相关的信息统计与报表输出，下面介绍一些常用功能的实现。

7.2.1 查询全部职工信息

用户信息查询包括全部查询与分类查询，分类查询根据输入的条件对查询结果进行分类。本节实现所有职工信息的查询，并按职工入职时间对查询结果进行升、降序排列。

1. 信息显示视图

打开 Zend Studio 集成开发环境，在 ZF Tool 工具窗口中输入命令：

```
zf create action list User 1 admin
```

在 admin 模块的 User 控制器中添加 list 方法，修改 application\modules\admin\views\scripts\list.phtml 视图文件：

```
<br/><br />
<style type="text/css">
…
</style>
<div class="nav">
    <ul>
        <li class="li1"><a href="#" class="a3">管理</a></li>
        <li class="li2"><a href="#" class="a3">查询</a></li>
    </ul>
</div>
<div class="cell_h"></div>
<div class="nav1">  <!—这是是页面位置提示 -->  </div>
<div class="cell_h"></div>
<div class="looktype">
    查看方式:<a href="#" class="a14">入职时间降序</a>
  <a href="#" class="a14">入职时间升序</a>
</div>
<div class="cell_h"></div>
<table class="table" align="center">
    <tr>
        <th style="…">职工姓名</th>
        <th style="…">所属于部门</th>
        <th style="…">籍贯</th>
        <th style="…">入职时间</th>
        <th style="…">E-mail</th>
        <th style="…">查看</th>
        <th style="…">编辑</th>
        <th style="…">删除</th>
    </tr>
<tr>
        <td style="#">     </td>
        <td style="#">     </td>
        <td style="#">     </td>
        <td style="#">     </td>
        <td style="#">     </td>
        <td style="#">
<a href="#"><img src="/images/list.gif" title="查看" border="0"/></a>
</td>
        <td style="#">
<a href="#"><img src="/images/edit.gif" title="编辑" border="0"/></a>
</td>
        <td style="#">
<a href="#"><img src="/images/del.gif" title="删除" border="0"/></a>
</td>
    </tr>
</table>
<div class="cell_h"></div>
<div class="nav1">   <!—这里是页面导航 -->   </div>
<div class="cell_h"></div>
```

该代码中省略了 CSS 样式代码，请大家查询教材源码。页面效果如图 7.2 所示。



图 7.2　用户信息显示页面效果

2. 添加方法代码

在 admin 模块的 list 方法中编写代码：

```
public function listAction ()
{
      //获取页面参数
      $page = $this->_request->getParam('page');
      $lt = $this->_request->getParam('lt');
      //每页显示记录数
       $pageSize = 8;
       //查询结果分页
       $paginator = $this->_user->getByPage($lt, $page, $pageSize);
       //传递页面参数
       $this->view->lt = $lt;
       $this->view->paginator = $paginator;
       //设置页面信息
       $title = '职工信息列表';
       $this->_setPageInfo($title, $title);
  }
```

3. 修改 User 模型

打开 application\models\user.php 文件，添加 getByPage 方法，代码如下：

```
public function getByPage ($orderIndex, $page = 1,
                              $itemCountPerPage = 10, $pageRange = 5)
{
          //生成 select 对象
          $select = $this->select();
          $select->from($this->_name);
          //排序方式
          if ($orderIndex == 0) {
               $order = 'regtime desc';
          } elseif ($orderIndex == 1) {
               $order = 'regtime';
```

```
            }
            $select->order('usertype desc');
            $select->order($order);
            //实例 Zend_Paginator 对象
            $paginatorAdapter = new Zend_Paginator_Adapter_DbSelect($select);
            $paginator = new Zend_Paginator($paginatorAdapter);
            $paginator->setPageRange($pageRange)->setItemCountPerPage(
                                $itemCountPerPage)->setCurrentPageNumber($page);
            return $paginator;
}
```

4. 添加页面设置函数

```
private function _setPageInfo ($title, $position, $layout = 'admin')
{
        //当前页面位置
  $this->view->position = $this->_config['pageInfo']['default']['title']
                                        . ' >> 管理中心 >> ' . $position;
        //页面布局
        if ($layout == null) {
             $this->_helper->layout->disableLayout();
        } else {
             $this->_helper->layout->setLayout($layout);
        }
    }
```

5. 添加分页控制视图

由于浏览器窗口大小有限,用户信息的显示需要进行分页处理。在 application\modules\views\scripts 目录下添加 list_user_pagination_control.phtml 视图文件,并编写代码:

```
<div style="width:280px; height:20px; line-height:20px; color:#0257AE; font-weight:
bold; text-align:center; border:1px solid #92C7ED;
background-color:#FAFEFF; margin-right:8px; float:left;">
每页  <font style="color:#999999; font-family:Arial; font-size:12px;"><?php echo
$this->itemCountPerPage?></font> 条/共  <font style="color:#999999; font-
family:Arial; font-size:12px;"><?php echo $this->totalItemCount?></font> 条  第
 <font style="color:#999999; font-family:Arial; font-size:12px;"><?php echo
$this->current?></font> 页/共  <font style="color:#999999; font-family:
Arial; font-size:12px;"><?php echo $this->pageCount?></font> 页
</div>
<?php if($this->totalItemCount > 0):?>
    <?php if($this->current > 1):?>
    <div style="width:40px; height:20px; line-height:20px; border:1px solid #92C7ED;
background-color:<?php if($this->current == 1):?>#1A75D2;<?php else:?>#FAFEFF;<?php
endif;?> font-weight:bold; float:left; margin-right:8px;">
        <a href="<?php echo $this->baseUrl('/admin/user/list/lt/'.$this->lt.'/page/
1')?>" class="<?php if($this->current == 1):?>a4<?php else:?>a3<?php endif;?>">首页</a>
    </div>
    <?php endif;?>
    <?php foreach ($this->pagesInRange as $page):?>
    <div style="width:20px; height:20px; line-height:20px; border:1px solid #92C7ED;
```

```
background-color:<?php if($this->current == $page):?>#1A75D2;<?php else:?>#FAFEFF;
<?php endif;?> font-family:Arial; font-style:italic; font-weight:bold; float:left; margin-
right:8px;">
            <a href="<?php echo $this->baseUrl('/admin/user/list/lt/'.$this->lt.'/page/
'.$page)?>" class="<?php if($this->current == $page):?>a4<?php else:?>a3<?php endif;?
>"><?php echo $page?></a>
      </div>
      <?php endforeach;?>
      <?php if($this->pageCount > $this->pageRange && $this->pageCount!= $this->
current):?>
      <div style="width:40px; height:20px; line-height:20px; border:1px solid #92C7ED;
background-color:<?php if($this->current == $this->last):?>#1A75D2;<?php else:?>
#FAFEFF;<?php endif;?> font-weight:bold; float:left; margin-right:8px;">
            <a href="<?php echo $this->baseUrl('/admin/user/list/lt/'.$this->lt.'/page/
'.$this->last)?>" class="<?php if($this->current == $this->last):?>a4<?php else:?>
a3<?php endif;?>">尾页</a>
      </div>
      <?php endif;?>
<?php endif;?>
```

6. 实现所有职工信息查询

图 7.2 所示的是用户信息显示空模板页面，如果要显示查询内容，还必须在模板文件中添加内容，代码如下：

```
<br/><br />
<style type="text/css">
…
</style>
<div class="nav">
      <ul>
            <li class="li1">
    <a href="/admin/user/list/flag/all/lt/0/page/1" class="a3">管理</a>
            </li>
            <li class="li2"><a href="#" class="a3">查询</a></li>
      </ul>
</div>
<div class="cell_h"></div>
<div class="nav1">
<?php echo $this->paginationControl($this->paginator,'Sliding',
'list_user_pagination_control.phtml',array('lt'=>$this->lt))?>
</div>
<div class="cell_h"></div>
<div class="looktype">
      查看方式：<a href="#" class="a14">入职时间降序</a>
   <a href="#" class="a14">入职时间升序</a>
</div>
<div class="cell_h"></div>
<table class="table" align="center">
      <tr>
            <th style="…">职工姓名</th>
            <th style="…">所属于部门</th>
```

```
            < th style = " … ">籍贯</th>
            < th style = " … ">入职时间</th>
            < th style = " … "> E - mail </th>
            < th style = " … ">查看</th>
            < th style = " … ">编辑</th>
            < th style = " … ">删除</th>
      </tr>
<?php foreach ( $ this - > paginator as  $ key = >  $ user) :?>
< tr >
            < td style = " # "><?php echo  $ user[ 'username'] </td>
            < td style = " # ">       </td>
            < td style = " # ">       </td>
            < td style = " # ">       </td>
            < td style = " # ">       </td>
            < td style = " # ">
< a href = " # ">< img src = "/images/list.gif" title = "查看" border = "0"/></a>
</td>
            < td style = " # ">
< a href = " # ">< img src = "/images/edit.gif" title = "编辑" border = "0"/></a>
</td>
            < td style = " # ">
< a href = " # ">< img src = "/images/del.gif" title = "删除" border = "0"/></a>
</td>
      </tr>
<?php endforeach;?>
</table>

< div class = "cell_h"></div>
< div class = "nav1">
          <?php echo  $ this - > paginationControl( $ this - > paginator, 'Sliding',
          'list_user_pagination_control.phtml',array('lt' = > $ this - > lt))?>
 </div>
< div class = "cell_h"></div>
```

为了让管理员打开 User 控制器首页就能看到职工的全部信息，在 admin 模块 User 控制器的 index 方法中添加如下页面转向代码：

```
public function indexAction()
{
       …
       $ this - > redirect('/admin/user/list');
 }
```

完成上述全部工作后，在浏览器中输入 http://wmoams.com/admin/user，即可看到如图 7.1 所示的页面效果。

7.2.2 职工信息的有序排列

这里只实现按职工入职时间对查询结果进行升、降序排列的功能。

查看上面 User 模型中的 getByPage 方法的定义，它的第 1 个参数 orderIndex 是用来设

定查询结果的排序方式的，而 list 方法中调用 getByPage 方法时带入的第 1 个参数是 lt，所以，只要在页面视图中根据不同的链接设定不同的 lt 参数值即可。

在 list.phtml 视图文件中添加如下代码：

```
<div class = "looktype">
    查看方式：<a href = "/admin/user/list/lt/0/page/1" class = "a14">
                注册时间降序</a>
            <a href = "/admin/user/list/lt/1/page/1" class = "a3">
              注册时间升序</a>
</div>
```

7.2.3 职工信息的条件查询

这里只实现按职工所属部门的条件查询。查询中采用模糊查询的方式，即用户可以只在查询表单中输入"所属部门"中的部分关键词就可以进行查询。例如，要查询所有"环境工程学院"的职工，在查询表单中输入"环境"即可。

1. 添加方法

打开 Zend Studio 集成开发环境，在 ZF Tool 工具窗口中输入命令：

```
zf create action search User 1 admin
```

在 admin 模块的 User 控制器中添加 search 方法。添加代码如下：

```
public function searchAction()
{
    //获取页面参数
    $keywords = urldecode( $this -> getRequest() -> getParam('keywords'));
    //开始查询并返回查询结果
    if ( $keywords != null && trim( $keywords) != '') {
        //查询表单回填数据
        $fData = array();
        $fData['keywords'] = $keywords;
        $this -> view -> fData = $fData;
        $users = $this -> _employee -> getByLike( $keywords);
        $this -> view -> users = $users;
        $this -> view -> isShow = 'T';
    }
    //设置页面信息
    $title = '用户信息查询';
    $this -> _setPageInfo( $title, $title);
}
```

2. 修改页面视图

打开 application\modules\admin\views\scripts\user\search.phtml 视图文件，将原有代码全部删除，并添加如下代码：

```
<br/><br />

…
```

```
<div class="cell_h"></div>
<div class="searchArea">
    <form name="form_searchProgram" id="form_searchProgram"
            method="post" action="
                <?php echo $this->baseUrl('/admin/user/search')?>"
                style="margin:0px; padding:0px;"
                onsubmit="return chkInputSearchProgram()">
    <div class="cell_h"></div>
    <div class="p1">
                关键字:<input type="text" name="keywords" id="keywords"
  class="input" style="width:300px; height:17px; line-height:17px;"
            <?php if($this->fData!=null):?> value="
            <?php echo $this->fData['keywords']?>"<?php endif;?>/>
            <input type="submit" value="查询" />
    </div>
    </form>
</div>
<?php if($this->isShow=='T'):?>
<div class="cell_h"></div>
<table class="table" align="center">
…
```

代码中的省略部分与前面的 list.phtml 视图文件相同。需要特别强调的是，search.phtml 的删除"链接"的 href 值为：

```
<?php echo '/admin/user/delete/id/'.$user['id'].'/isSearch/T/keywords/\'+encodeURI
(\''.$this->fData['keywords'].'\')+\''
```

在查询表单输入关键字后的页面效果如图 7.3 所示。



图 7.3　用户信息条件查询

7.2.4 职工信息的添加

管理系统中的用户注册分为批量注入与单个用户注册两种。系统运行前，由系统管理员通过文件一次性注入职工信息，在平常的管理中只需添加一些少量的特殊用户。

1. 单个添加

1）创建 admin 模块的 User 模型

打开 Zend Studio 集成开发环境中的 ZF Tool 工具，在命令窗口中输入：

```
zf create model User admin
```

在 admin 模块中创建 User 模型，并添加代码：

```
class Admin_Model_User extends Zend_Db_Table
{
    protected $_name = 'tb_users';
    protected $_primary = 'id';
    public function addByInfo($data = array()){
        $row = $this->createRow();
        if(count($data) > 0){
            foreach ($data as $key => $value){
                $row->$key = $value;
            }
            $row->save();
            return $row->id;
        }
        else{
            return null;
        }
    }
}
```

上述代码在 User 模型中添加了一个名为 addByInfo 的方法，该方法完成职工信息的数据库插入。注意，Zend Tool 工具生成的模型类的名称为 Admin_Model_User，它的前缀与默认 default 中的 Wm_不同。如果要让系统在使用该类时能够自动搜索，需要在 admin 模块中添加启动文件 Bootstrp.php，该文件位于 application\modules\admin\目录下。

```
class Admin_Bootstrap extends Zend_Application_Module_Bootstrap
{
  public function init(){

  }
}
```

注意，其基类与默认模块中的启动类基类不同。

2）创建控制器方法

在 admin 模块的 User 控制器中添加 register 方法，并添加代码：

```
public function registerAction()
{
```

```
        $this->view->isShow = 'F';
        if($this->getRequest()->getParam('admin')){
            $this->view->isShow = 'T';
            //$this->render();
        }
        $formUser = new Wm_Form_User();
        $formUser->removeElement('netname');
        …
        $formUser->removeElement('role');
        $formUser->getElement('submit')->setLabel('提交');

        if($this->getRequest()->isPost()){
            if($formUser->isValid($_POST)){
                $userDate = $formUser->getValues();
                $modelUser = new Admin_Model_User();
                $userDate['password'] = md5($userDate['password']);
                $newUser = $modelUser->addByInfo($userDate);
                if($newUser){
                    $this->redirect('/admin/user/detail/id/'.$newUser);
                }else{
                    throw new Zend_Exception('注册用户失败!');
                    exit();
                }
            }
        }
        $this->view->formUser = $formUser;
        //设置页面信息
        $title = '职工个人注册';
        $this->_setPageInfo($title,$title);
    }
```

3）修改视图

打开 register 对应的视图文件 register.phtml，编写代码：

```
…
<div class="cell_h"></div>
<div class="searchArea">
    <form name="form_searchProgram" id="form_searchProgram" method="post" action="/admin/user/register" style="margin:0px; padding:0px;" onsubmit="return chkInputSearchProgram()">
    <div class="cell_h"></div>
    <div class="p1">
  注册操作员：<input type="text" name="admin" id="keywords" class="input" readonly style="width:300px; height:17px; line-height:17px;" value="<?php echo $this->user->getIdentity()->username ?>" />    <input type="submit" value="确认" />
    </div>
    </form>
</div>
<?php if($this->isShow=='T'):?>
<div class="cell_h"></div>
<table class="table" align="center">
    <tr>
```

```
            < th style = "width:30px;height:24px; border:1px solid #BDE2FD;
                        background:url('/images/title_bg1.gif');">标签</th>
            < th style = "height:24px; border:1px solid #BDE2FD;
                          background:url('/images/title_bg1.gif');">内容</th>
            < th style = "width:250px; height:24px; border:1px solid #BDE2FD;
                            background:url('/images/title_bg1.gif');">说明</th>
        </tr>
        < tr>
            < td colspan = "2" style = " border:1px solid #BDE2FD;
                        text - align:left;"><?php echo $this - > formUser; ?></td>
            < td style = "height:250px; border:1px solid #BDE2FD;
                                                text - align:left;">  </td>
        </tr>
    </table>
    <?php endif;?>
    < div class = "cell_h"></div>
```

职工信息添加页面效果如图 7.4 所示。这里没有处理表单输入错误时的提示信息，请参照第 6 章的相关章节。



图 7.4　用户注册

4) 职工详细信息

职工信息添加成功后，页面跳转到该职工的详细信息页面，可以在这里校对其全部信息。

在 admin 模块的 User 控制器中添加 detail 方法，并修改视图，效果如图 7.5 所示。

在 Zend Studio 集成开发环境中打开 application\modules\admin\controllers 目录下的 User 控制器文件，在 detail 方法中添加如下代码：

```
public function detailAction()
{
```

```
        $id = $this->getRequest()->getParam('id');
        …
        $formUser->removeElement('submit');
        $employee = $this->_employee->find($id)->current();
        $employee = $employee->toArray();
        $formUser->populate($employee);
        $this->view->formUser = $formUser;
        $this->view->employee = $employee['username'];
        //设置页面信息
        $title = '职工详细信息';
        $this->_setPageInfo($title, $title);
    }
```

打开 application\modules\admin\views\scripts\user 目录下的 detail.phtml 视图文件，并编写代码：

```
<br /><br />
<h2 align="center"><?php echo $this->employee?>   ----   个人详细信息</h2>
<hr />
<?php echo $this->formUser;?>
```



图 7.5　用户详细信息

2. 批量添加

职工信息的批量添加通过文件来完成。为了系统信息的安全与系统操作的可核查，在批量添加过程中，文件被选定后，系统默认将文件上传至服务器。为了简便，这里只实现由 Microsoft Office Excel 提供的职工信息登记文件。

1）创建方法

在 admin 模块的 User 控制器中添加 insert 方法，并添加代码：

```
public function insertAction()
{
        $addNum = 1;
        $addLastid = 0;
        $this->view->isShow = 'F';
        if ($this->getRequest()->isPost()){
            $adapter = new Zend_File_Transfer_Adapter_Http();
            $path = APPLICATION_PATH.'/resources/
                                  '.date('Y-m-d').'/userInformation/';
    $folder = new Zend_Search_Lucene_Storage_Directory_Filesystem($path);
            $fileInfo = $adapter->getFileInfo();
            $extName = 'csv';
            $filename = 'admin'.$this->_user->getIdentity()->id.time().'.'.
            $extName;
            $adapter->addFilter('Rename',array(
                            'target'=>$filename,'overwrite'=>true));
            $adapter->setDestination($path);
            $adapter->addValidator('Extension',false,array('csv'));
            if($adapter->receive()){
                $file = $path.$filename;
            }else{
                throw new Zend_Exception('文件上传失败!');
                exit();
            }

            $modeUser = new Admin_Model_User();
            if (($handle = fopen($file,'r')) !== FALSE) {
                $columns = fgetcsv($handle,1000,",");
            }
            while (($data = fgetcsv($handle,1000,",")) !== FALSE) {
                $info = array_combine($columns,$data);
                $addLastid = $modeUser->addByInfo($info);
                $addNum++;
            }
            fclose($handle);
            $select = $this->_employee->select();
            $select->where('id>?',$addLastid-$addNum-1);
            $select->limit($addNum);
            $users = $modeUser->fetchAll($select);
            $users = $users->toArray();
            $this->view->users = $users;
            $this->view->isShow = 'T';
        }
        //设置页面信息
        $title = '职工批量注册';
        $this->_setPageInfo($title,$title);
    }
```

以上代码实现了文件上传、数据的读取与插入等一系列功能，当然可以用函数分成多个功能模块，但为了大家学习方便，我们把它们放在了一个函数中。对于代码的含义将在下一章中详细介绍，这里先把效果做出来。

2）修改视图

打开 application\modules\admin\views\scripts\user\insert.phtml 视图文件，将代码修改为：

```
…
<div class = "nav">
    <ul>
        <li class = "li2">
            <a href = "/admin/user/list/flag/all/lt/0/page/1"
                      class = "a3">信息管理</a>
        </li>
        <li class = "li2">
            <a href = "/admin/user/search" class = "a3">信息查询</a>
        </li>
        <li class = "li2">
            <a href = "/admin/user/register" class = "a3">个人注册</a>
        </li>
        <li class = "li1">
            <a href = "/admin/user/insert" class = "a3">批量注册</a>
        </li>
    </ul>
</div>
<div class = "cell_h"></div>
<div class = "searchArea">
    <form name = "form_insert" id = "form_insert" onsubmit =
                "javascript:return window.confirm(
                            '确认注入该文件中的职工信息吗?')?true:false"
method = "post" enctype = "multipart/form - data"
 action = "/admin/user/insert" style = "margin:0px;padding:0px;">
    <div class = "cell_h"></div>
    <div class = "p1">
  请选择文件:<input type = "file" name = "fileName" id = "file" class = "input" style = "width:
300px; height:20px; line - height:20px;" />
   <input type = "submit" style = "font - size: 10px; width:60px;height:20px;
line - height:20px;padding:0" value = "确定" />
          </div>
    </form>
</div>
…
```

批量添加注册文件时、数据注入完成后的效果分别如图 7.6 和图 7.7 所示。



图 7.6　用户信息批量注入确认页面



图 7.7　用户信息批量注入完成页面

7.2.5　职工信息的删除

在 admin 模块的 User 控制器中添加 delete 方法，并编写代码：

```
public function deleteAction()
{
    //页面编码
    header('content-type:text/html; charset=utf-8');
    //去掉视图和布局
    $this->_helper->layout()->disableLayout();
    $this->_helper->viewRenderer->setNoRender();
    //接收页面参数
    $id = $this->getRequest()->getParam('id');
    $isSearch = $this->getRequest()->getParam('isSearch');
```

```
        $select = $this->_employee->select();
        $select->where('id = ?', $id);
        $result = $this->_employee->fetchRow($select);
        $result->delete();
        //返回
        if ($isSearch != null && $isSearch == 'T') {
            $this->_redirect('/admin/user/search/keywords/' .
                    urlencode($this->_request->getParam('keywords')));
        } else {
            $this->_redirect('/admin/user/list/lt/0/page/1');
        }
        exit();
    }
```

删除职工信息不需要跳转到另外一个页面，只需要把结果在当前页面上显示出来就可以了。也就是说，删除操作完成后要能看到记录从页面上消失。

注意：删除操作可能是在全部信息显示页面；也可能是在查询页面，删除操作完成后，页面类型不能改变。所以，在 list. phtml 和 search. phtml 页面中的“删除”链接其 href 值是不一样的，请大家特别注意。

7.3 用户信息的前台管理

用户进入系统主页后，可以对自己的信息进行适当管理，包括查看、更改部分信息等。下面就来完成这方面的功能。

7.3.1 创建用户信息面板

当用户通过登录认证后，进入系统前台主页，在该页面中应该显示用户的相关信息，例如职工姓名、所属部门、角色等。如果用户具有进入系统后台管理中心的权限，还应该显示进入管理中心的链接。另外，用户注销的链接也是必不可少的。

打开 Zend Studio 集成开发环境，双击打开项目 wmProject 的 application\layouts\header. phtml 视图文件，添加代码：

```
<?php
   $auth = Zend_Registry::get('auth');
   $user = $auth->getIdentity();
   switch ($user->role){
       case 'admin': $role = '系统管理员';break;
       case 'edit': $role = '部门管理员';break;
       default: $role = '普通员工';break;
   }
?>
<div id = "header">
    <div id = "logo">
        <img src = "/images/logo.jpg">
        <div id = "usepanel">
            今天是：<?php echo date('Y 年 m 月 d 日',time()) ?>   
            欢迎您：<a href = '/user/account/id/<?php echo $user->id ?>'>
                    <?php echo $user->username ?></a>   
```

```
            所属部门：<?php echo $user->department ?>   
            您是：<?php echo $role ?>    
            <?php if($user->role == 'admin'):?>
            <a href = '/admin/user/list'>
                        进入系统后台管理中心</a>    
            <?php endif;?>
            <a href = '/user/logout'>注销</a>
        </div>
    </div>
</div>
```

第 1～9 行代码通过注册表对象 auth 取出用户认证信息，并存放在 User 中，接着将用户角色用通俗的术语输出；其后的代码表示用户信息面板，这里显示当前日期、用户姓名(职工姓名)、所属部门以及用户角色，可以根据具体情况对显示内容做适当的调整。代码中用户姓名和所属部门是从登录认证信息中直接取出的，并且给用户姓名添加了超级链接，让用户单击自己的姓名可以查看个人详细信息。

页面效果如图 7.8 所示。



图 7.8 用户信息面板效果图

7.3.2 个人信息主页的设计

当用户单击用户信息面板中的姓名或选择“个人办公”|“个人名片”菜单项时，页面跳转到用户个人信息主页，在这里可以查看个人全部信息，也可以对某些信息进行修改。

完成后的效果如图 7.9 所示。



图 7.9 用户个人详细信息

1. 创建控制器方法

为 User 控制器添加 account 方法。打开 Zend Studio 集成开发环境中的 ZF Tool 工具窗口，输入命令：

```
zf create action account User
```

注意：操作是在默认模块中进行的，User 后面的模板名 default 可以省略。在 account 方法中添加代码：

```
public function accountAction()
    {
        $ id = $ this->getRequest()->getParam('id');
        if(! $ id){
            $ id = $ this->_user->getIdentity()->id;
        }
        $ modelUser = new Wm_Model_User();
        $ user = $ modelUser->getUser( $ id);
        $ this->view->user = $ user;
    }
```

上述代码中的 if 语句是考虑用户以选择“个人办公”|“个人名片”菜单项的方式进入个人信息主页的情况。

代码中的_user 是在 User 控制器中定义的变量，用于存放用户认证信息。其定义及初始化代码如下：

```
class UserController extends Zend_Controller_Action
{
    protected $ _user = null;
    public function init()
    {
        $ auth = Zend_Registry::get('auth');
        if( $ auth->hasIdentity()){
            $ this->_user = $ auth;
        }
    }
  …
}
```

2. 创建用户模型

在 User 控制器的 account 方法中通过传递用户 id 利用用户模型的 getUser 方法从数据库中查询到登录用户的全部信息，所以需要创建用于处理业务逻辑的 User 模型。请大家注意，我们在编程的过程中始终是围绕 M-V-C 的模式进行的，当然顺序有时会不一样，抓住这个关键点，思路会更清晰一些。

在 Zend Studio 集成开发环境的 ZF Tool 工具窗口中输入命令：

```
zf create model User
```

在 User 模型中添加代码：

```
class Wm_Model_User extends Zend_Db_Table
{
    protected $ _name = 'tb_users';
```

```
    public function getUser( $ where)
    {
        if (is_numeric( $ where)){
            $ row = $ this -> find( $ where) -> current();
        }
        if (is_array( $ where)){
            $ select = $ this -> select();
            if (count( $ where) > 0){
                foreach ( $ where as $ key => $ value){
                    $ select -> where( $ key. " = ?", $ value);
                }
            }
            $ row = $ this -> fetchRow( $ select);
        }
        if ( $ row){
            return $ row;
        }
        else{
            return null;
        }
    }
}
```

上述代码定义了一个根据条件查询用户信息的 getUser 方法，当传入的参数为数值时，说明是用主键直接查找，对于 find 方法的使用已在第 5 章中做了详细的介绍。对数据库的查询操作可以使用不同的函数，但大家要注意各个函数的返回值的类型差异，这关系到数据在视图中的输出。

3. 设计视图页面

从图 7.7 所示的页面效果可以看到，用户个人信息页面分为上、下两个区，上区为操作区，显示了当前用户能进行的操作；下区为信息详细显示区，采用了选项卡的页面形式，将信息进行了分类，分别是基本信息、工作信息、联系信息以及其他信息。

选项卡页面的实现要结合 CSS 样式表与 JavaScript 来完成，由于篇幅的关系，我们不做详细介绍，请大家查询源码。视图部分代码如下：

```
<div class = "menu3">
    <ul>
        <li id = "three1" onmouseover = "setTab('three',1,4)" class = "hover">
            基本信息</li>
        <li id = "three2" onmouseover = "setTab('three',2,4)">工作信息</li>
        <li id = "three3" onmouseover = "setTab('three',3,4)">联系信息</li>
        <li id = "three4" onmouseover = "setTab('three',4,4)">其他信息</li>
     </ul>
     <div class = "clear"></div>
 </div>
 <div class = "con_t" id = "con_three_1" >
      <table class = "info">
            <tr>
                <th  width = "15 % ">标签</th>
```

```
                <th width="35%">内容</th>
                <th width="15%">标签</th>
                <th width="35%">内容</th>
            </tr>
            <tr><td>姓名: </td>
                <td><?php echo $this->user['username']?></td>
                <td>年龄: </td>
                <td><?php echo $this->user['age']?></td>
            </tr>
            <tr><td>性别: </td>
                <td><?php echo $this->user['sex']?'男':'女'?></td>
                <td>民族: </td>
                <td><?php echo $this->user['nation']?></td>
            </tr>
              …
        </table>
    <div class="clear"></div>
</div>
```

7.3.3 个人信息的修改

用户个人可以对某些信息进行修改,例如密码、联系方式、个人简介等。用户信息的修改使用表单来完成,当然还是用之前创建的 user 表单而无须新建。

1. 修改表单

个人信息中的用户照片是通过文件上传得到的。为了在信息修改页面也能看到用户照片,我们将第 6 章创建的 user 表单做一下调整,代码如下:

```
//用户照片
$photo = $this->createElement('image','photo');
$photo->setLabel('照片: ');
$photo->setAttrib('src','/images/nophoto.gif');
$this->addElement($photo);
//照片上传
$avatar = $this->createElement('file','avatar');
$avatar->setLabel('照片上传: ');
$avatar->addValidator('Size',false,2048000);
$avatar->addValidator('Count',false,1);
$avatar->setRequired(false);
$this->addElement($avatar);
```

由于 Zend_Form 表单的 file 表单元素需要使用 File 装饰器来修饰,所以要在表单文件的后面加上文件表单元素装饰器代码:

```
$this->setElementDecorators(array(
…
));
$this->getElement('avatar')->setDecorators(
        array(
            'File',
```

```
        array(array('data' => 'HtmlTag'),array('tag' => 'td')),
        array('Label',array('tag' => 'th')),
        array(array('row' => 'HtmlTag'),array('tag' => 'tr'))
        )
    );
$this->setDecorators(array(
…
));
```

请务必注意添加代码的位置。

2. 添加方法

在 Zend Studio 集成开发环境的 ZF Tool 工具窗口中输入命令：

```
zf create action update User
```

在默认模块的 User 控制器中创建 update 方法，编写代码：

```
public function updateAction()
{
    $auth = Zend_Registry::get('auth');
      $id = $auth->getIdentity()->id;

      $formUser = new Wm_Form_User();
      …
      $formUser->removeElement('avatar');
      //修改"登录"按钮的标签为"提交"
      $formUser->getElement('submit')->setLabel('提交');
      $modelUser = new Wm_Model_User();
      $message = '';
      if($this->getRequest()->isPost()){
          if ($formUser->isValid($_POST)){
              $formData = $formUser->getValues();
              $validUser = $modelUser->validUser($formData['netname']);
          //系统中的登录名不能相同
              if(!$validUser){
                  $formData['password'] = md5($formData['password']);
                  unset($formData['password2']);
                  unset($formData['photo']);
                  $where = 'id = '.$id;
                    $modelUser->update($formData, $where);
              }else{
                  $message .= '对不起,系统中已经存在同名用户!';
              }
          }else{
          //错误提示信息的提取
          $arrayMessage = $formUser->getMessages();
          if (isset($arrayMessage['netname']['0'])) {
              $message .= '  '.$arrayMessage['netname']['0'].'<br />';
          }
          …
      }
```

```
        }
        $user = $modelUser->find($id)->current();
        $formUser->populate($user->toArray());
        if($user['avatar']){
            $formUser->getElement('photo')->setAttrib('src',$user['avatar']);
        }
        $this->view->user = $user;
        $this->view->formUser = $formUser;
        $this->view->message = $message;
    }
```

3. 修改视图

打开 application\views\scripts\user\update.phtml 视图文件，添加代码：

```
<br /><br />
<?php
  echo "<h3>".$this->user->username."个人信息</h3>";
  echo $this->formUser;
?>
```

在图 7.7 所示的个人信息主页中单击上区中的“更改”按钮，转向个人信息修改页面，效果如图 7.10 所示。



图 7.10 用户信息的更改

4. 处理表单

1) 验证提示信息的显示

在 update.phtml 中添加代码，当验证失败时，在表单的下面用红色集中显示错误提示信息。

```
<br /><br />
<?php
```

```
    echo "< h3 >". $ this -> user -> username. "个人信息修改</h3 >< hr />";
    echo $ this -> formUser;
?>
< div style = "color: # ff0000"><?php echo $ this -> message ?></div >
```

修改 User 控制器的 update 方法，取出错误信息。

```
if (isset( $ arrayMessage[ 'netname'][ '0'])) {
                    $ message . = '   '. $ arrayMessage[ 'netname'][ '0']. '< br />';
                }
                if (isset( $ arrayMessage[ 'phone'][ '0'])) {
                    $ message . = '   '. $ arrayMessage[ 'phone'][ '0']. '< br />';
                }
                if (isset( $ arrayMessage[ 'email'][ '0'])) {
                    $ message . = '   '. $ arrayMessage[ 'email'][ '0']. '< br />';
                }
                if (isset( $ arrayMessage[ 'avatar'][ '0'])) {
                    $ message . = '   '. $ arrayMessage[ 'avatar'][ '0'];
                }
```

2）避免用户登录名同名

用户登录系统采用的是登录名，也就是所谓的昵称。用户可以根据自己的喜好来设置昵称，但在整个系统中它必须是唯一的。所以，当用户修改昵称时，应判断是否与系统中已注册用户的登录名同名。

在 User 模型中添加方法，打开 application\model\User. php 模型文件，添加验证方法及代码：

```
public function validUser( $ netname)
{
        $ select = $ this -> select();
        $ select -> where("netname = ?", $ netname);
        $ result = $ this -> fetchRow( $ select);
        if ( $ result -> netname == $ netname){
            return $ result -> id;
        }
        else{
            return FALSE;
        }
}
```

在 User 控制器的 update 方法中添加代码，完成同名验证：

```
if ( $ formUser -> isValid( $ formData)){
            $ validUser = $ modelUser -> validUser( $ formData[ 'netname']);
            if(! $ validUser){
                        ;
                    }else{
                        $ message . = '对不起，系统中已经存在同名用户！';
                    }

                }else{
…
}
```

7.4 忘记密码功能的实现

如果用户忘记了登录密码该如何处理呢？这是系统设计中不得不考虑的问题。从前面的登录处理过程中我们知道，要得到旧密码是比较困难的，因为存入数据库中的密码都是经过 MD5 加密处理的。我们把这个经过加密的密码字符串返回给用户，他们需要使用工具破解之后才能得到原来的密码，显然这种方法是不实用的。更为方便可行的方法是，通过注册时的电子邮箱发给用户一个重置的新密码，它由系统自动生成的随机字符组成，用户使用这个新密码登录系统后可以自行修改成容易记忆的密码。

处理流程：用户需要先通过一个表单提交用户名和邮箱，我们对此用户名及邮箱进行查询，如果数据库中有该用户，则系统自动生成一个随机密码，发送到这位用户的注册邮箱；如果没有该用户，返回相应的提示。下面通过这个功能的实现向大家介绍 Zend Framework 中电子邮件的发送方法。

1. 添加链接

在登录页面上添加一个"忘记密码"的链接。打开 application\views\scripts\user\login.phtml 视图文件，添加如下代码：

```
<div id="login_center">
        …
        <div style="position: absolute;right:300px;top:400px">
            <a href="/user/reset-password">忘记密码</a>
        </div>
</div>
```

2. 创建方法

在 Zend Studio 集成开发环境的 ZF Tool 工具窗口中输入命令：

```
zf create action resetpassword User
```

在默认模块的 User 控制器中添加 resetpassword 方法，并编写代码：

```
public function resetpasswordAction()
{
        $this->_helper->layout->disableLayout();
        $message = "请输入登录名及您的注册邮箱，重置的密码会发送到该邮箱！
                <br /><br />用新密码登录后，进入个人信息页面，更改密码！";
        $formUser = new Wm_Form_User();
        $formUser->removeElement('username');
        …
        $formUser->removeElement('role');
        $formUser->getElement('submit')->setLabel('密码重置');
        if ($this->getRequest()->isPost()){
            if ($formUser->isValid($_POST)){
                $modelUser = new Wm_Model_User();
                $where = array(
                        'netname' => $formUser->getValue('netname'),
```

```
                'email' => $ formUser -> getValue('email')
        );
        $ user = $ modelUser -> getUser( $ where);
        if ( $ user){
            $ newPassword = $ this -> _makePassword(6);
            $ where = 'id = '. $ user -> id;
            $ modelUser -> update(array('password' =>
                                md5( $ newPassword)), $ where);
            set_time_limit(0);                  //不限时间,默认时间为 30 秒
          $ mailTransport = new Zend_Mail_Transport_Smtp(
                    'smtp.163.com',
                    array( 'auth' => 'login',
                    'username' => 'w111111',       //邮箱用户名
                    'password' => '19666666',      //邮箱密码
                    'ssl' => 'ssl'));
            $ mail = new Zend_Mail('utf - 8');
         $ mail -> setBodyHtml('<p>您的密码被系统重置为: . $ newPassword.
              '<br />建议您在下次登录后重新更改为容易记忆的新密码。</p>');
            $ mail -> setSubject('您在微梦办公自动化管理系统的新密码');
            $ mail -> setFrom('wenpingwei@163.com','微梦工作室');
            $ mail -> addTo( $ user -> email, $ user -> username);
            $ mail -> send( $ mailTransport);
            $ this -> redirect('/');
        }
    }else{
        $ arrayMessage = $ formUser -> getMessages();
        …
    }
  }
  $ this -> view -> formUser = $ formUser;
  $ this -> view -> message = $ message;
}
```

上述代码中用到的自定义函数 makePassword 产生 6 位随机密码。代码如下:

```
protected function _makePassword( $ length)
  {
    $ possible = "0123456789abcdefghijklmnopqrstuvwxyz
                                ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    $ str = "";
    while(strlen( $ str) < $ length){
        $ str . = substr( $ possible,(rand() % strlen( $ possible)),1);
    }
    return( $ str);
  }
```

3. 修改视图文件

依照登录界面的形式设计密码重置界面,效果如图 7.11 所示。

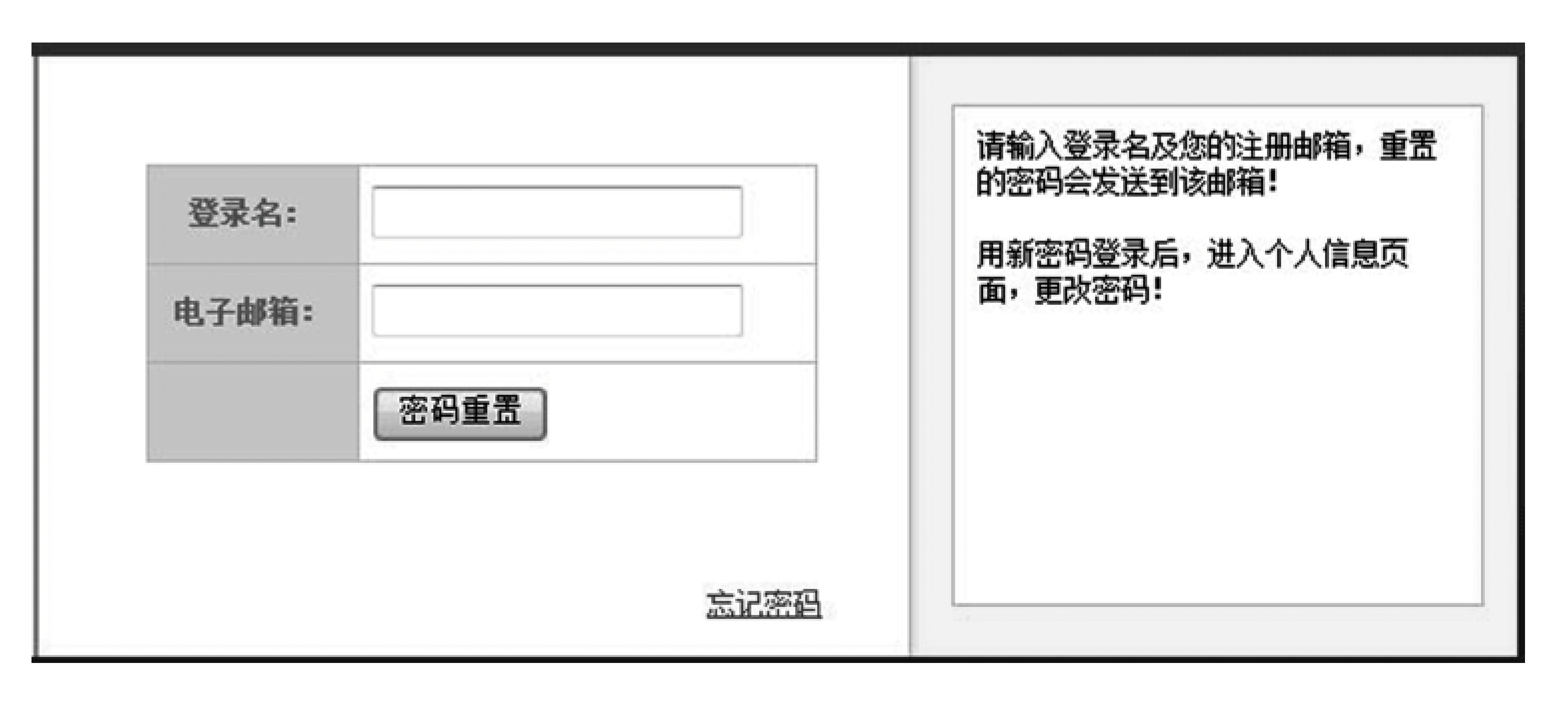


图 7.11　用户密码的重置

7.5　本 章 小 结

本章通过用户信息管理功能的实现实例，介绍了 Zend Framework 框架的模块技术，以及数据库的增、删、改、查等详细操作方法，既是对第 5 章内容的进一步阐述与扩充，也是对前面其他章节知识的实际应用，做到了温故而知新。

学习本章应重点掌握 Zend Framework 的自定义模块的创建方法、目录结构及其与默认模块的关系。对于数据库的操作，除了要熟练掌握常用的基本操作外，还要重点掌握文中所介绍的通过文件批量注入数据的方法。另外，用户密码重置中的邮件操作也是值得关注的一个知识点。

第 8 章　公文信息管理模块

公文管理是办公自动化管理系统的核心功能，是实现企业信息化管理的基础。公文的种类有很多，包括通知、决定、报告、纪要、请示、函等。这些公文的内容、缓急程序以及保密级别虽然有所不同，但结构基本上是相似的，可以用一个统一的数据模型对它们进行管理。

本章实现系统公文的管理功能，包括公文的拟稿、校核、查询、删除等，以此介绍 Zend Framework 的分页显示组件 Zend_Paginator、文件组件 Zend_File 的使用。

8.1　功 能 预 览

办公自动化系统中的公文是企业办公过程中信息交流的载体，对它的管理要求做到规范、准确、快捷与安全。规范性是指公文的拟稿、审核、发布、收件、办结等流程要符合企业管理规范；准确性是指公文送达目标的精准；快捷则要求系统功能要能满足用户及时获取、保存公文信息的便利。

由于教材篇幅以及写作重点的关系，下面只实现一些公文管理的基本功能。

（1）系统主页的公文显示，如图 8.1 所示。



图 8.1　公文在系统主页上的显示效果

（2）公文管理主页上的分类显示，如图 8.2 所示。



图 8.2　公文在系统主页上的显示效果

（3）个人公文管理页面效果，如图 8.3 所示。



图 8.3　个人公文管理页面效果

（4）公文拟稿页面效果，如图 8.4 所示。



图 8.4　公文拟稿页面效果

8.2　数据库设计

数据库是系统信息永久保存的方式之一，关系到信息的查询、系统的运行效率等诸多方面。为了简便，本系统公文采用统一的数据表进行管理。

根据第 3 章的总体设计及业务逻辑的需要，公文信息管理数据表由“公文信息”表、“部门信息”表和“公文类型”表组成。

1. 公文信息表

打开 phpMyAdmin 数据库管理工具，在 db_wmoams 数据库中新建名为 tb_docs 的数据表，其字段含义见表 8.1，字段属性如图 8.5 所示。

表 8.1　tb_docs 数据表的字段说明

字　段　名	说　　明
id	自增变量，主键，公文在系统中的唯一标识
fileno	公文编号
uid	拟稿人
checkid	审核人
title	公文标题
keyword	关键词
typeid	公文类型 ID
body	公文内容
attach	公文附件
fromdepart	发文部门
todepart	收文部门

续表

字 段 名	说 明
status	公文状态
createtime	发布时间
endtime	公文处理截止时间
suggest	是否允许意见反馈
secret	保密级别
urgency	公文缓急等级
tododepart	公文待办部门
doingdepart	公文在办部门
donedepar	公文办结部门

名字	类型	排序规则	属性	空	默认	额外
id	int(10)		UNSIGNED	否	无	AUTO_INCREMENT
fileno	varchar(20)	utf8_general_ci		是	NULL	
uid	int(10)		UNSIGNED	是	0	
checkid	int(10)		UNSIGNED	是	NULL	
title	varchar(255)	utf8_general_ci		是	NULL	
keyword	varchar(20)	utf8_general_ci		是	NULL	
typeid	int(10)		UNSIGNED	是	NULL	
body	longtext	utf8_general_ci		是	NULL	
attach	varchar(100)	utf8_general_ci		是	NULL	
fromdepart	int(10)		UNSIGNED	是	NULL	
todepart	int(10)		UNSIGNED	是	NULL	
status	tinyint(4)			是	1	
createtime	int(11)			是	0	
endtime	int(11)			是	NULL	
suggest	tinyint(4)			是	NULL	
secret	tinyint(4)			是	0	
urgency	tinyint(4)			是	0	
tododepart	varchar(100)	utf8_general_ci		是	NULL	
doingdepart	varchar(100)	utf8_general_ci		是	NULL	
donedepart	varchar(100)	utf8_general_ci		是	NULL	

图 8.5 tb_docs 数据表的属性设置

2. 部门信息表

在 db_wmoams 数据库中新建名为 tb_depart 的数据表，其字段含义见表 8.2，字段属性如图 8.6 所示。

表 8.2 tb_depart 数据表的字段说明

字 段 名	说 明
id	自增变量，主键，部门在系统中的唯一标识
dno	部门编号
fid	部门所属单位 ID
name	部门名称
file_prefix	部门发文前缀

名字	类型	排序规则	属性	空	默认	额外
id	int(10)		UNSIGNED	否	无	AUTO_INCREMENT
dno	char(20)	utf8_general_ci		否	无	
fid	int(10)		UNSIGNED	否	无	
name	varchar(50)	utf8_general_ci		否	无	
path	varchar(255)	utf8_general_ci		否	无	
file_prefix	char(20)	utf8_general_ci		是	NULL	

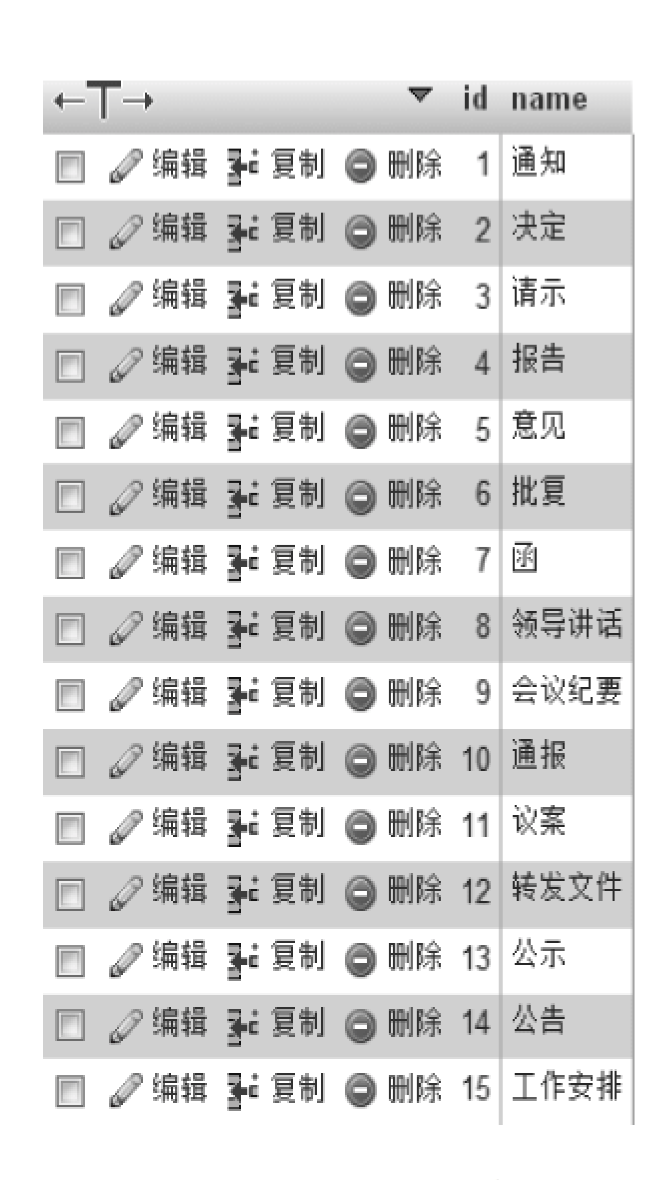
图 8.6　tb_depart 数据表的属性设置

3. 公文类型表

在 db_wmoams 数据库中新建名为 tb_doctype 的数据表，其字段含义见表 8.3。本系统设置的公文类型如图 8.7 所示。

表 8.3　tb_doctype 数据表的字段说明

字　段　名	说　　明
id	自增变量，主键，公文类型在系统中的唯一标识
name	公文类型名称



←T→				id	name
☐	编辑	复制	删除	1	通知
☐	编辑	复制	删除	2	决定
☐	编辑	复制	删除	3	请示
☐	编辑	复制	删除	4	报告
☐	编辑	复制	删除	5	意见
☐	编辑	复制	删除	6	批复
☐	编辑	复制	删除	7	函
☐	编辑	复制	删除	8	领导讲话
☐	编辑	复制	删除	9	会议纪要
☐	编辑	复制	删除	10	通报
☐	编辑	复制	删除	11	议案
☐	编辑	复制	删除	12	转发文件
☐	编辑	复制	删除	13	公示
☐	编辑	复制	删除	14	公告
☐	编辑	复制	删除	15	工作安排

图 8.7　系统公文类型

8.3　公文信息显示

系统公文的输出，在不同的页面上应该具有不同的侧重点。在系统主页上主要显示最新发布的公文信息，如图 8.1 所示，让用户进入系统就能及时了解最新的各种信息动态；而在“公文管理”菜单模块的页面上显示公文内容时应进行适当的分类，让用户能方便、快捷地找到与自己相关的内容。

8.3.1 模型与控制器的创建

1. 创建模型

打开 Zend Studio 集成开发环境，选择 wmProject 项目下的任何一个文件或文件夹，通过菜单打开 ZF Tool 工具窗口，输入命令：

```
zf create model Page
```

这里是在默认模块 default 中创建模型，Page 模型名的后面省略了模块名。

打开 application\models\Page.phtp 模型文件，给模型类添加基类 Zend_Db_Table，并添加对应的数据库表 tb_docs 和主键字段 id，这样，模型 Wm_Model_Page 就变成了一个数据库的表模型。如果以 id 为主键，这里的主键设置是可以省略的，就像我们前面做的一样，但如果数据表的主键字段名不为 id，就可以用这种方式进行设置。设置了主键，可以方便地用 Zend_Db 组件的类方法 find 进行查询。

模型代码如下：

```
class Wm_Model_Page extends Zend__Db_Table
{
  protected $_name = 'tb_docs';
  protected $_primary = 'id';
}
```

上面创建模型采用的是 ZF Tool 工具，这样做的好处是能自动生成目录结构并创建 PHP 文件，并且模型类名是按照 Zend Framework 的规则生成的，可以自动加载。大家也可以手动添加目录及类文件，如果不使用像上面那样的类名，在程序中使用时需要添加 include 或 require 等相关语句或函数，将模型类引入到文件中。

2. 创建控制器

在集成开发环境的 ZF Tool 工具窗口中输入命令：

```
zf create controller Document
```

在默认模块中创建 Document 控制器。通过 ZF Tool 工具添加控制器不仅可以自动生成目录结构，还可以自动添加 index 方法以及对应的 index.phtml 文件。

ZF Tool 工具在完成上述工作的同时还在项目 wmProject 根目录中生成了一个名为 .zfproject.xml 的文件，该文件记录了项目的资源名称及目录。如果要删除某个资源，例如要删除刚创建的控制器 Document，在删除控制器类文件、视图文件及文件夹的同时还需要删除该文件中有关 Document 小节的内容，否则，Document 这个名字就不能再使用了，因为它已经被记录在资源里。.zfproject.xml 文件是在项目开发过程中使用的，项目完成后可以将其删除。下面是该文件中的部分代码：

```
<controllerFile controllerName = "Document">
      <actionMethod actionName = "index"/>
      …
</controllerFile>
…
<viewsDirectory>
```

```
    <viewScriptsDirectory>
        <viewControllerScriptsDirectory forControllerName="Document">
            <viewScriptFile forActionName="index"/>
        </viewControllerScriptsDirectory>
    </viewScriptsDirectory>
<viewHelpersDirectory/>
```

8.3.2 系统主页公文的列表显示

从如图 8.1 所示的系统主页界面可以看出，这里的公文是以列表的形式输出的，并且只显示公文标题。为了方便查看，每篇公文的标题均设置了超级链接，单击链接可以跳转到该公文的详细内容页面，以了解更多的内容。由于主页幅面有限，显示的内容不可能太多，所以，在每个功能区另外设置 more 链接，通过它用户可以查询到更多的相关信息。

1. 添加模型方法

打开 application\models\Page.php 模型文件，添加 getPages 方法，该方法用于从数据表中获取多条满足条件的记录。代码如下：

```
class Wm_Model_Page extends Zend_Db_Table
{
    protected $_name = 'tb_docs';
    protected $_primary = 'id';
    public function getPages ($orderIndex, $where = null,
                              $page = 1, $itemCountPerPage = 10, $pageRange = 5)
    {
        //生成 select 对象
        $select = $this->select();
        $select->where('typeid = ?', $where);
        //排序方式
        if ($orderIndex == 0) {
            $order = 'createtime desc';
        } elseif ($orderIndex == 1) {
            $order = 'createtime';
        }
        $select->order('urgency desc');
        $select->order($order);
        //实例 Zend_Paginator 对象
        $paginatorAdapter = new Zend_Paginator_Adapter_DbSelect($select);
        $paginator = new Zend_Paginator($paginatorAdapter);
        $paginator->setPageRange($pageRange)->
                    setItemCountPerPage($itemCountPerPage)->
                    setCurrentPageNumber($page);
        return $paginator;
    }
}
```

由于完成的功能相似，上述代码与第 7 章用户管理模块中的 User 模型的代码基本上是相同的，实际上可以用一些技术方案使代码共享。这里为了不使问题复杂化，重新定义了一个新的模型类。

上述代码中使用了 Zend Framework 的 Zend_Paginator 进行信息的分页显示，在第 7 章中我们已经领略到了它的方便与简捷，下面详细了解一下这个类及其常用方法。

2. Zend_Paginator 组件

在 Web 开发过程中，分页是一项非常麻烦的工作，页面之间涉及很多的参数需要传送，而且应用中一般都存在着大量的列表页面，在每个页面需要重复编写同样的代码很多次，实在是一项烦琐的苦差事。Zend Framework 为用户提供了一个非常简便的分页解决方案。

使用 Zend_Paginator 组件进行分页需要分 3 个步骤来实施：

1）设置数据适配器

为了把数据集在页面上进行分页显示，Zend_Paginator 必须首先获取到数据。Zend_Paginator 支持几种常见的适配器，例如 Array、DbSelect、DbTableSelect 等。Array 适配器使用一个 PHP 数组；DbSelect 适配器使用一个 Zend_Db_Select 实例，得到一个数组；DbTableSelect 适配器使用一个 Zend_Db_Table_Select 实例，得到一个 Zend_Db_Table_Rowset_Abstract 实例。这里使用的是 DbTableSelect 适配器。

在上述代码中，先根据各类条件构建了一个 Zend_Db_Table_Select 实例 select，然后用 select 作为参数实例化 Zend_Paginator_Adapter_DbSelect 类，这样就得到了一个包含分页数据的页面对象。注意，此时的对象已不全是从数据库中查询到的记录集，里面包含了大量的分页信息。

2）设置页面参数

在分页对象创建好之后，还要给它设置一些必要的页面参数，例如每页显示多条数据、分页显示控制条中最多显示多少页、当前是第几页等，这些都是通过 Zend_Paginator_Adapter_DbSelect 类的方法来完成的。

在上述代码中使用了 setPageRange 方法设置控制条的页码范围，默认为 5，即分页控制条中最多允许显示 5 个页码，超出的页码要通过“上一页”、“下一页”或“首页”、“尾页”查看；使用 setItemCountPerPage 方法设置每页显示的数据数量，默认为 10 条；另外，还使用 setCurrentPageNumber 方法设置了当前页码，默认为第 1 页。

3）创建分页显示控制条视图文件

在 Web 应用中有各种各样的分页显示控制条，这里采用第 7 章中的分页视图文件。这个文件是 Zend Framework 使用手册例程中的代码经过适当修改后得到的，请大家参考官方手册。

在分页显示控制条视图文件编写好之后，把它放在 application\views\scripts 文件夹下 application\views\scripts\partials 目录下或 application\views\scripts 目录下的自建文件夹下均可，只要在调用时注意路径即可。application\views\scripts\partials 文件夹是在第 4 章中自建的文件夹，里面已经存放了一个页面控制视图文件，把分页控制条视图文件 list_page_Pagination_control.phtml 也放在该目录下。

分页控制条视图文件准备好之后，在页面视图中加入如下代码，显示分页控制条。

```
<?php echo $this->paginationControl($this->paginator,
        'Sliding','partials/list_page_pagination_control.phtml',
?>
```

代码中的 paginator 为分页适配器对象，Sliding 为分页控制条中页码的滚动方式，第 3

个参数便是分页控制条视图文件，我们存放它于 partials 目录下，所以带上了文件夹名。

3. 添加控制器方法代码

在第 4 章中已经完成了系统主页页面的设计，下面只需要在 Index 控制器的 main 方法中添加代码，调用 page 模型的 getPages 方法获取数据就可以了。

```
public function mainAction()
{
    …
    $where = 1;                                              //通知公示类型的公文
    $docPages1 = $this->_page->getPages(1, $where,1,8);
    $this->view->docPages1 = $docPages1;
    $where = 2;                                              //最新其他公文
    $docPages2 = $this->_page->getPages(1, $where,1,8);
    $this->view->docPages2 = $docPages2;
    …
}
```

根据主页页面大小，设置每个功能区最多显示 8 条数据、取第 1 页的内容。这里只是示意性地验证了程序的正确性，具体的业务逻辑还需要用户根据具体情况进行设计。每个功能区中显示的数据也可以任意设置，主页页面可以在高度方向自由伸缩。

代码中的_page 是自定义的 Index 控制器变量，并在 Index 的 init 方法中初始化为模型 Wm_Model_Page 的实例对象。如下所示：

```
class IndexController extends Zend_Controller_Action
{
   …
    protected $_page = null;
    public function init()
    {
       …
      $this->_page = new Wm_Model_Page();
    }
```

4. 添加视图代码

在 Index 控制器的 main 方法中得到的页面内容通过 docPage 变量传递到了对应的 main. phtml 视图中，需要在该视图文件中添加代码，在适当的位置将公文标题显示出来。

打开 application\views\scripts\index\main. phtml 视图文件，添加如下代码：

```
…
<div class="center_box">
    <div id="item1">
        …
        <ul>
            <?php foreach ($this->docPages as $key => $page):?>
            <li><img alt="" src="/images/xing.gif">  
                <a href="#"><?php echo $page['title']?></a></li>
            <?php endforeach;?>
        </ul>
    </div>
    …
```

系统主页上不需要显示分页控制条，不必添加渲染控制条的代码，页面效果如图 8.8 所示。



图 8.8　系统主页公文信息的显示

8.3.3　公文信息的详细显示

在系统主页的各个功能区中，列表显示的公文标题都设置了超级链接，用户单击标题进入公文信息的详细显示页面。

1. 添加控制器方法

公文信息的详细显示由 Document 控制器管理。打开 Zend Studio 集成开发环境，在 ZF Tool 工具窗口中输入命令：

```
zf create action docdetail Document
```

在 Document 控制器中创建 docdetail 方法，并添加如下代码：

```
public function docdetailAction()
{
        $id = $this->getRequest()->getParam('id');
        $docPage = $this->_page->getPage($id);
        $this->view->docPage = $docPage;
}
```

代码中的_page 是自定义的 Document 控制器变量，其定义与初始化和 8.3.2 节中 Index 控制器的相同。

2. 添加模型方法

在上述控制器方法中使用了 page 模型的 getPage 方法，下面编写该方法的代码：

```
public function getPage($where = null)
  {
        $select = $this->select()->setIntegrityCheck(false);
        $select->from('tb_docs','*');
        if (is_numeric($where)){
             $select->where("tb_docs.id = ?", $where);
        }
        if (is_array($where) & count($where) > 0){
             foreach ($where as $key => $value){
                  $select->where($key.'= ?', $value);
             }
```

```
    }
    $ select -> join('tb_users', 'tb_docs.uid = tb_users.id', 'username as editname');
    $ select -> join('tb_users as tb_check', 'tb_docs.checkid = tb_check.id', 'username as checkname');
    $ select -> join('tb_depart', 'tb_docs.fromdepart = tb_depart.id', 'name as fromname');
    $ row = $ this -> fetchRow( $ select);
    if( $ select){
        return $ row;
    }
    else {
        return null;
    }
}
```

上述 getPage 方法用来获取公文的详细信息，这里采用了 Zend_Db_Select 的联合查询方式。

所谓“联合查询”，就是获取的信息分别来自多个相互关联的数据表。请大家查看数据库中的 tb_docs 表，它的 uid、checkid、fromdepart 等字段的数据类型均为 int。这就是说字段里存储的是数值，分别代表“拟稿人”、“发布人”及“发布部门”的 id。但是在视图中，这些字段是不能以数字的形式输出的，而要显示各数字所代表的“名字”。例如，在输出“拟稿人”的时候要通过 tb_docs 表中字段 uid 的值查找 tb_uses 表中的 id，得到 tb_uses 表中的字段 name，也就是这个人的名字。

代码中的 join 函数是在 Zend_Db_Select 类中执行联合查询的方法，调用该方法要带 3 个参数。其中，第 1 个参数是联合表名；第 2 个参数是联合查询的条件；第 3 个参数是查询的字段。对于上述代码来说，联合查询执行完毕后，得到的记录除主表 tb_docs 中的全部字段外，还添加了 editname、checkname 以及 fromname 3 个字段，所以在下面的视图中通过这些字段可以获取到相应的名称。代码中的 as 是定义的别名。

3. 修改视图文件

打开 application\views\scripts\document\docdetail.phtml 视图文件，添加如下代码，显示公文的详细信息。

```
<style>
<!--
.title5{text-align:center; border-bottom:1px dashed #cacaca; padding:0 0 15px 0; margin-bottom:15px;}
.title4{ font-size:22px; text-align:center; line-height:50px; font-weight:600}
.format{margin-left:50px}
p{font-size:14px;   line-height:30px; }
-->
</style>
<br /><br />
<h2 class="title4"><?php echo $ this->docPage['title'];?></h2>
<div class="title5">
    <?php
        echo "<span>发布部门：". $ this->docPage['fromname']."</span>";
        echo "<span class='format'>发布时间：".
            date('Y-m-d', $ this->docPage['createtime'])."</span>";
```

```
        ?>
    </div>
    <p><?php echo $this->docPage['body'];?></p>
    <hr />
    <?php
        echo "<span>文件编号：". $this->docPage['fileno']."</span>";
        echo "<span class = 'format'>拟稿人：".
                        $this->docPage['editname']."</span>";
          echo "<span class = 'format'>发布人：".
                      $this->docPage['checkname']."</span>";
    ?>
    <hr />
```

请大家注意代码中的阴影部分，这些都是通过联合查询从表 tb_user 和表 tb_depar 中得到的。代码添加完毕后，打开系统主页，单击“通知公告”展示区的通知标题，页面效果如图 8.9 所示。



图 8.9　公文信息的详细显示

8.3.4　全部公文信息的列表显示

在系统主页中只显示了最新发布的部分公文信息，下面实现在 Document 控制器中显示所有公文信息的功能，以方便用户查询。

1. 页面效果

一般全部信息的列表显示都配有相应的排序与条件查询功能，就像我们在第 7 章中所做的那样。这里为了简便，省略了查询部分，在显示全部信息的时候固定地进行了公文的排序与分类，如图 8.10 所示。

图 8.10 所示的信息显示页面分为上、下两个功能区，上区是操作功能区，这里会根据用

户的权限显示不同的功能按钮；下区为公文列表显示区，这里显示公文的编号、标题、发布部门及发布日期等信息，并配有分页控制条，对页面进行控制。



图 8.10 公文信息管理主页

2. 创建控制器方法

打开 Zend Studio 集成开发环境的 ZF Tool 命令窗口，输入命令：

```
zf create action list Document
```

在控制器 Document 中创建 list 方法，并添加如下代码：

```
public function listAction()
{
        //获取页面参数
        $page = $this->getRequest()->getParam('page');
        $lt = $this->getRequest()->getParam('lt');
        //每页显示记录数
        $pageSize = 7;
        //查询结果分页
        $paginator1 = $this->_page->getPages($lt,1, $page, $pageSize);
        $paginator2 = $this->_page->getPages($lt,2, $page, $pageSize);
        $paginator3 = $this->_page->getPages($lt,5, $page, $pageSize);
        $paginator4 = $this->_page->getPages($lt,8, $page, $pageSize);
        $paginator5 = $this->_page->getPages($lt,15, $page, $pageSize);
        $paginator = array($paginator1, $paginator2, $paginator3,
                          $paginator4, $paginator5);
        //传递页面参数
        $this->view->lt = $lt;
        $this->view->paginator = $paginator;
    }
```

这里根据公文的类别对信息进行了简单的分类，实际开发时，请大家根据项目要求进行

具体设计。该代码中使用了 Page 模型的 getPages 方法来获取查询到的分页数据。

在图 8.10 给出的效果中显示了公文信息的“发布部门”,所以模型方法中也需要采用数据库联合查询的方式从 tb_depart 表中提取“发布部门”的名称。

3. 修改模型方法

打开 application\models\Page.php 模型文件,修改如下阴影位置的代码:

```
public function getPages ( $orderIndex, $where = null, $page = 1, $itemCountPerPage = 10,
 $pageRange = 5)
{
        //生成 select 对象
        $select = $this->select()->setIntegrityCheck(false);
        $select->from($this->_name);
        $select->where('tb_docs.typeid = ?', $where);
        //排序方式
         if ($orderIndex == 0) {
            $order = 'tb_docs.createtime desc';
        } elseif ($orderIndex == 1) {
            $order = 'tb_docs.createtime';
        }
        $select->order('tb_docs.urgency');
        $select->order($order);
        $select->join('tb_depart',
                  'tb_docs.fromdepart = tb_depart.id', 'name as fromname');
        …
        return $paginator;
    }
```

4. 修改视图文件

打开 application\views\scripts\document\docdetail.phtml 视图文件,删除原有全部内容,添加如下代码:

```
<br /><br />
<style>
<!--
  …
 -->
</style>
<?php $typeid = 2?>
<div class = "operBox">
    <ul class = "operNav">
        <li><a href = "#"><img alt = "" src = "/images/queryBtn.jpg"></a></li>
        <li><a href = "/document/list">
                          <img alt = "" src = "/images/refreshBtn.jpg"></a></li>
    </ul>
</div>
<div class = "file">这里是公文列表,包括通知、公示、公告、批复、报告等
<hr />
 <div class = "tabbox">
        <div class = "menu3">
          <ul>
```

```
                <li id="three1" onmouseover="setTablist('three',1,5)"
                                    class="hover">会议通知</li>
                <li id="three2" onmouseover="setTablist('three',2,5)">
                                                  公示公告</li>
                <li id="three3" onmouseover="setTablist('three',3,5)">
                                                  启事通报</li>
                <li id="three4" onmouseover="setTablist('three',4,5)">
                                                  信息发布</li>
                <li id="three5" onmouseover="setTablist('three',5,5)" >
                                                  工作安排</li>
              </ul>
              <div class="clear"></div>
              </div>
              <div class="con_t" id="con_three_1" >
                 <table class="list">
                     <tr>
                         <th width="15%">文件编号</th>
                         <th  width="45%">标题</th>
                         <th  width="30%">发布单位</th>
                         <th  width="10%">发布日期</th>
                     </tr>
                     <?php foreach ($this->paginator[0] as $key=> $page):?>
                     <tr><td><?php echo $page['fileno'] ?></td>
                           <td><?php echo $page['title']?></td>
                           <td><?php echo $page['fromname']?></td>
                         <td><?php echo date('Y-m-d', $page['createtime'])?></td>
                      </tr>
                    <?php endforeach;?>
                 </table>
                 <div class="clear"></div>
              </div>
               …
   <div class="nav1" id="nav_three_1" style="display:block">
      <?php echo $this->paginationControl(
                  $this->paginator[0],'Sliding',
                  'partials/list_page_pagination_control.phtml',
                  array('lt'=>$this->lt))?>
 </div>
   …
 </div>
```

视图用 JavaScript 实现了选项卡形式的页面形式，在项目的 public\js 文件夹中有相应的源代码，请大家查阅。

在上述代码中，中间的阴影部分为数据的输出；末尾的阴影部分为分页控制条渲染代码。这两部分都只列出了一个小节的内容，省略的代码与此相同。

8.4 部门公文信息管理

8.3 节中实现的是对全部公文信息的显示，没有涉及部门与个人。其实在实际工作过程中，人们最关心的是与自己有关的公文信息。本节实现个人公文信息的管理，包括全部公

文、待办公文、在办公文和办结公文等。

8.4.1 部门公文信息管理流程

部门公文信息管理是由各部门指定的部门管理员完成的,该管理员具有比普通用户高的管理权限,可以签收、发布文件。

发给本部门的所有信息首先存放在“待办公文”表中,部门公文信息管理员查看确认后签收文件,即将文件转入“在办公文”表;在“在办公文”表中显示该文件办理的截止日期,提醒管理员合理安排工作时间;公文处理完毕,并得到本部门领导或发文部门认可后,就可以将该文件由“在办公文”表转入“办结公文”表中。

在表 8.1 所示的字段描述中,数据表 tb_docs 中设置的 3 个字段(tododepart、doingdepart 和 donedepart)分别用来记录上述公文状态的变化。例如序号为“1”的公文发给了“计算机学院”,当计算机学院的管理员查询公文时,就会在“待办公文”表中发现它,管理员确认签收后,就会在 tb_docs 表的 doingdepart 字段中存入计算机学院的序号(tb_depart 表中计算机学院的 id 值,假设为 6),此时 tb_docs 表的 tododepart 字段中的“6”被删除;公文处理完毕,管理员将该文件移入“办结公文”中,即执行 tb_docs 表的 doingdepart 字段中的“6”的删除、在 donedepart 字段中添加“6”的操作。这样,通过搜索这 3 个字段中的关键字就可以知道该文件在各个签收部门的状态。

如图 8.11 所示表示发给“计算机学院”的全部公文,包括待办公文、在办公文、已办公文。



文件编号	标题	发布单位	发布日期
计科2014080321	计算机科学与技术学院学生优秀作品展示	计算机学院	2014-08-03
计科2014080318	清华大学出版社教材立项通知	计算机学院	2014-08-03
计科2014080223	关于顾兴等同志任职的公示	计算机学院	2014-08-03
计科2014080223	关于申报湖北省科技厅科技项目的通知	计算机学院	2014-08-02
教2014073101	数据结构教学	教务处	2014-06-19
教2014073101	微梦工作室	教务处	2014-06-03
人事2014073101	微信平台讲座	人事处	2014-05-29

图 8.11 部门全部公文信息页面

如图 8.12 所示表示发给“计算机学院”的所有待处理的公文,界面中的操作为“签收”链接。



图 8.12　部门待办公文信息页面

8.4.2　部门公文接收的实现

1. 添加模型方法

接收部门公文信息就是以该部门名称为关键字,在数据库中进行查询。所以,需要在Page模型中添加方法,以关键字为参数实现数据库的查询。

打开模型文件 application\models\Page.php,添加如下代码:

```
public function getByLike ( $field, $department, $page = 1, $itemCountPerPage = 8, $pageRange = 5)
    {
        $select = $this->select()->setIntegrityCheck(false);
        $select->from( $this->_name);
        $select->where( $this->_name.'.'. $field.' like ?','%'. $department .'%');
        $order = 'tb_docs.createtime';
        $select->order( $order);
        $select->join('tb_depart','tb_docs.fromdepart = tb_depart.id','name as fromname');
        $result = $this->fetchAll( $select);
        if ( $result != null) {
            $result = $result->toArray();
        }
        //实例 Zend_Paginator 对象
        $paginatorAdapter = new Zend_Paginator_Adapter_Array( $result);
        $paginator = new Zend_Paginator( $paginatorAdapter);
        $paginator->setPageRange( $pageRange)->setItemCountPerPage(
                    $itemCountPerPage)->setCurrentPageNumber( $page);
        return $paginator;
    }
```

这里采用了模糊查询的方法,并且可以针对不同的字段进行查询。代码中的分页适配

器用的是 Array 适配器，请大家注意。

2. 添加控制器方法

打开 Zend Studio 集成开发环境的 ZF Tool 工具窗口，输入命令：

```
zf create action receive Document
```

在 Document 控制器中添加 receive 方法，这也是导航菜单项“公文管理”|“收文管理”的链接。编写代码：

```
public function receiveAction()
{
        //获取页面参数
        $page = $this->getRequest()->getParam('page');
        $lt = $this->getRequest()->getParam('lt');
        //每页显示记录数
        $pageSize = 7;
        //待办公文
        $field = 'tododepart ';
        $depart = $this->_user->getIdentity()->depart_id;
        $paginator1 = $this->_page->getByLike(
                                    $field, $depart, $page, $pageSize);
        …
        $paginator = array($paginator1, $paginator2, $paginator3,
                                        $paginator4, $paginator5);
        //传递页面参数
        $this->view->lt = $lt;
        $this->view->paginator = $paginator;
    }
```

代码中省略的部分为“在办公文”、“办结公文”代码，与“待办公文”的相似，只是查询的字段不同而已。

3. 修改视图文件

Document 控制器的 receive 方法对应的视图文件为 application\views\scripts\document\receive.phtml。它与 list 方法的视图基本相同，但要注意的是，分页控制条视图文件要重新复制一份，因为翻页链接这里对应的是 receive 方法。

8.5 公文文档的创建与发布

公文信息可以供所有用户浏览，但公文的发布需要相应的权限。关于用户权限，将在后面的第 12 章中详细介绍。公文的发布一般分为 3 个步骤，首先由拟稿人编辑公文文档，然后提交校核人校对，最后由领导批准发布。

8.5.1 公文信息表单

打开 Zend Studio 集成开发环境的 ZF Tool 工具窗口，输入命令：

```
zf create form File
```

创建 File 表单文件，它位于 application\forms 目录下，然后打开该文件，并在表单类 Wm_Form_File 的 init 初始化方法中添加代码：

```
class Wm_Form_File extends Zend_Form
{
public function init()
    {
        $this->setName('form_file')                    //表单名称
            ->setMethod('post')                        //表单提交方法
            ->addAttribs(array(
                        'style' => 'margin:10px'       //表单样式
        ));
        //文件编号
        $fileNo = $this->createElement('text','fileno');
        $fileNo->setLabel('文件编号：');
        $fileNo->setRequired(TRUE);
        $fileNo->setOptions(array('readonly' => true));
        $this->addElement($fileNo);
        //文件标题
        $title = $this->createElement('text','title');
        $title->setLabel('文件标题：');
        $title->setRequired(TRUE);
        $title->setOptions(array('style' =>'width:400px'));
        $title->addValidator('stringLength',false,array(4,100));
        $title->addErrorMessage('标题应有 2-50 个汉字.');
        $this->addElement($title);
        //文件类型
        $doctype = $this->createElement('select','typeid');
        $doctype->setLabel('文件类型：');
        $doctype->setRequired(TRUE);
        $modelType = new Wm_Model_Ftype();
        $ftypes = $modelType->getFtypes();
        if ($ftypes != null ){
            foreach($ftypes as $value){
                $doctype->addMultiOption($value->id, $value->name);
            }
        }
        $this->addElement($doctype);
        //发文部门
        $fromdepart = $this->createElement('text','fromdepart1');
        $fromdepart->setLabel('发文部门：');
        $fromdepart->setOptions(array('readonly' => true));
        $this->addElement($fromdepart);
        //拟稿人
        $eidt = $this->createElement('text','edit');
        $eidt->setLabel('拟稿人：');
        $eidt->setOptions(array('readonly' => true));
        $this->addElement($eidt);
        //收文部门
        $depart = $this->createElement('select','tododepart1');
```

```
$ depart -> setLabel('收文部门：');
$ depart -> setRequired(TRUE);
$ depart -> setOptions(array('onChange' =>'setDepart()'));
$ modelDeparts = new Wm_Model_Depart();
$ departs = $ modelDeparts -> getDeparts();
if ( $ departs != null ){
    foreach( $ departs as $ value){
        $ depart -> addMultiOption( $ value -> id, $ value -> name);
    }
}
$ this -> addElement( $ depart);

//收文部门
$ todepart = $ this -> createElement('text', 'tododepart');
$ todepart -> setLabel('');
$ todepart -> setOptions(array('readonly' => true, 'id' =>'todepart', 'style' =>'width:
400px'));
$ this -> addElement( $ todepart);
//文件内容
$ body = $ this -> createElement('textarea', 'body');
$ body -> setLabel('文件内容：');
$ body -> setAttribs(array('rows' => 10, 'cols' => 75));
$ body -> setRequired(TRUE);
$ this -> addElement( $ body);
//关键词
$ tags = $ this -> createElement('text', 'keyword');
$ tags -> setLabel('关键词：');
$ this -> addElement( $ tags);
//附件
$ attach = $ this -> createElement('file', 'attach');
$ attach -> setLabel('附件：');
$ attach -> setRequired(false);
$ this -> addElement( $ attach);
//发布状态
$ status = $ this -> createElement('checkbox', 'status');
$ status -> setLabel('是否直接发布：');
$ status -> setValue(0);
$ this -> addElement( $ status);
//是否可以反馈
$ allow = $ this -> createElement('checkbox', 'suggest');
$ allow -> setLabel('允许反馈：');
$ allow -> setValue(0);
$ this -> addElement( $ allow);
//提交按钮
$ submit = $ this -> createElement('submit', '提交');
$ this -> addElement( $ submit);
//表单装饰器
$ this -> setElementDecorators(array(
    'ViewHelper',
    'Errors',
    array(array('data' =>'HtmlTag'),array('tag' =>'td')),
```

```
            array('Label',array('tag'=>'th')),
            array(array('row'=>'HtmlTag'),array('tag'=>'tr')),
            ));
        $this->getElement('attach')->setDecorators(
            array(
                'File',
                array(array('data'=> 'HtmlTag'),array('tag'=> 'td')),
                array('Label',array('tag'=> 'th')),
                array(array('row'=> 'HtmlTag'),array('tag'=> 'tr'))
            )
        );
        $this->setDecorators(array(
        'FormElements',
        array('HtmlTag',array('tag'=>'table','class'=>'sheet')),'Form'
        ));
    }
}
```

表单页面效果如图 8.13 所示。

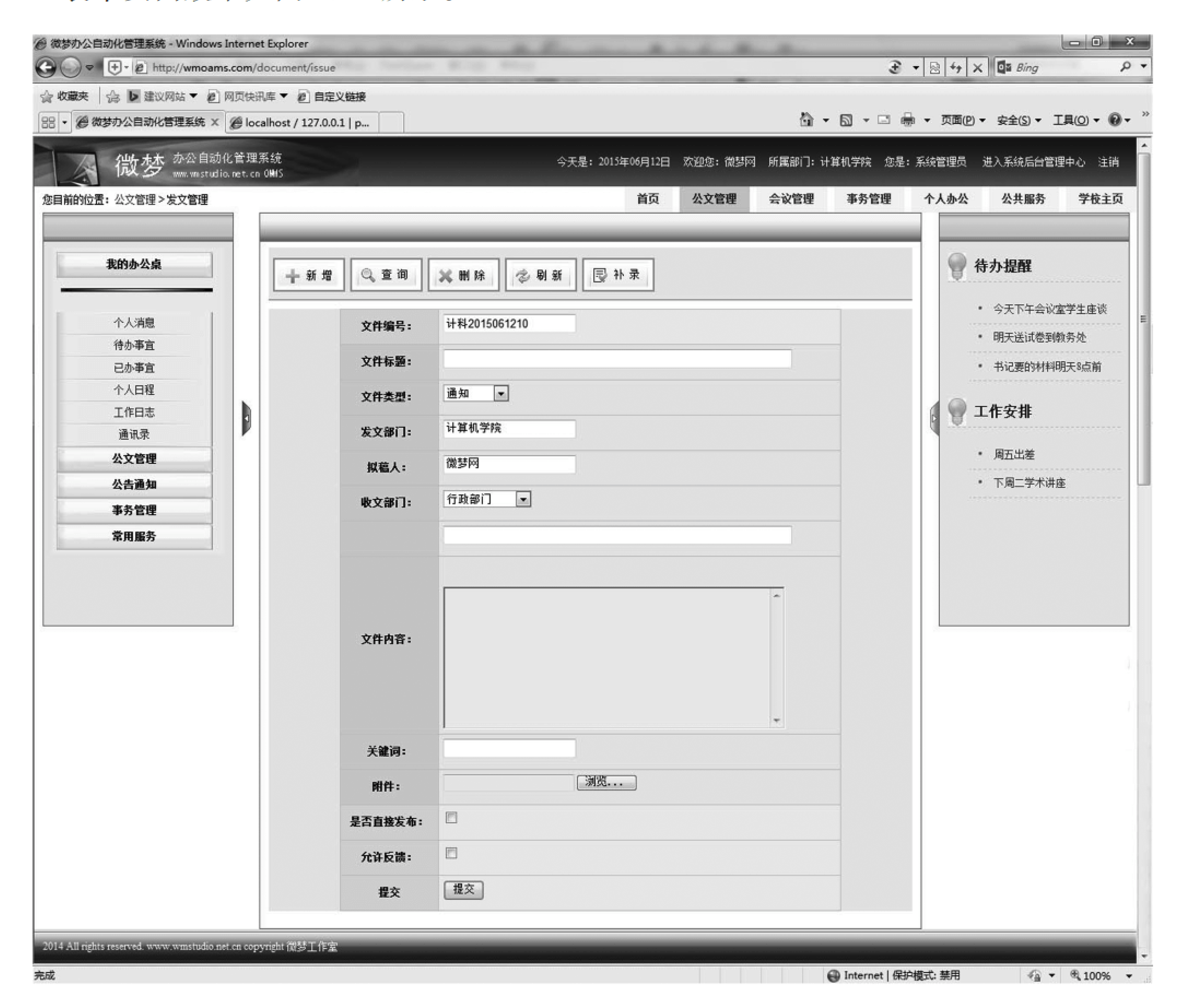


图 8.13　公文信息录入表单

8.5.2 公文表单处理方法

公文表单提交后，由 Document 控制器的 issue 方法进行处理。

打开 Zend Studio 集成开发环境的 ZF Tool 工具窗口，输入命令：

```
zf create action issue Document
```

在 Document 控制器中创建 issue 方法，打开控制器文件，编写如下代码：

```
public function issueAction()
{
        $formFile = new Wm_Form_File();
        $modelDepart = new Wm_Model_Depart();
        $depart_id = $this->_user->getIdentity()->depart_id;
        $fromDepart = $modelDepart->getDepart($depart_id);
        //生成文件编号
        $file_prefix = $fromDepart->file_prefix;
        $fileno = $file_prefix.date('YmdH');
        $formFile->getElement('fileno')->setValue($fileno);
        //发文部门
        $departName = $fromDepart->name;
        $formFile->getElement('fromdepart1')->setValue($departName);
        //拟稿人
        $edit = $this->_user->getIdentity()->username;
        $uid = $this->_user->getIdentity()->id;
        $formFile->getElement('edit')->setValue($edit);
        if($this->getRequest()->isPost()){
            if($formFile->isValid($_POST)){
                $data = $formFile->getValues();
                //移除不需要的表单元素
                unset($data['fromdepart1']);
                unset($data['tododepart1']);
                unset($data['edit']);
                unset($data['attach']);
                //设置其他字段值
                $data['uid'] = $uid;
                $data['checkid'] = $uid;
                $data['fromdepart'] = $depart_id;
                $data['createtime'] = time();
                //完成公文的添加
                $this->_page->addPage($data);
            }
        }
        $this->view->formFile = $formFile;
    }
```

这里，公文的文件编号、发文部门、拟稿人 3 项内容由程序自动设置。文件编号由发文部门的“文件前缀”加上“日期的年月日小时”组成，发文部门的文件前缀从数据库的 tb_depart 表中获取；“发文部门”与“拟稿人”直接设置为登录用户的相关信息，也就是说，公文的拟稿人就是已通过认证的登录用户本人。

8.5.3 公文信息处理模型方法

在上述控制器方法代码中使用模型方法 addPage 执行公文信息的添加，下面编写该方法的代码。

打开模型文件 application\models\Page. php，添加如下代码：

```
public function addPage( $ data = array())
{
        $ row = $ this -> createRow();
        if (count( $ data) > 0){
            foreach ( $ data as  $ key => $ value){
                $ row -> $ key = $ value;
            }
            $ row -> save();
            return $ row -> id;
        }
        else{
            throw new Zend_Exception('向数据库中写入数据失败!');
        }
}
```

这段代码比较简单，首先使用 Zend_Db_Table_Abstract 类的 createRow 方法创建一个 Zend_Db_Table_Row_Abstract 类的实例 row，然后通过 Zend_Db_Table_Row_Abstract 类的 save 方法将数据写入数据库中。

8.5.4 添加视图

上面创建的表单，在 issue 控制器方法中实例化后要在视图中渲染出来。打开视图文件 application\views\scripts\document\issue. phtml，删除原有全部内容，添加如下代码：

```
<br /><br />
<div class = "operBox">
    <ul class = "operNav">
        <li><a href = "/document/issue">
<img alt = "" src = "/images/addBtn. jpg"></a></li>
        <li><a href = "#"><img alt = "" src = "/images/queryBtn. jpg"></a></li>
        <li><a href = "#"><img alt = "" src = "/images/deleteBtn. jpg"></a></li>
        <li><a href = "/document/ issue ">
<img alt = "" src = "/images/refreshBtn. jpg"></a></li>
        <li><a href = "#"><img alt = "" src = "/images/editBtn. jpg"></a></li>
    </ul>
</div>
<div class = "file" ><?php echo $ this -> formFile?></div>
<script type = "text/javascript" >
function setDepart(){
    var depart1 = document. getElementById('todepart');
    var depart2 = document. getElementById('tododepart1');
    depart1. value = depart1. value + depart2. options[depart2. selectedIndex]. text + ",";
}
```

```
</script>
```

请大家注意上述代码的阴影部分，第 1 行是对公文信息表单的渲染，下面是一段 JavaScript 代码。

为什么要编写这样一个 JavaScript 函数呢？从图 8.13 给出的表单效果可以看出，“收文部门”文本框中记录的接收部门是由其上面的 select 选择表单元素提供的，拟稿人通过选择确定公文送达的部门。拟稿人的选择要及时地在“收文部门”文本框中显示出来，这段 JavaScript 代码就是用来完成这个功能的。其中的表单元素 id 和选择表单元素的 onChange() 属性都是在表单对象中设置的。

8.6 公文附件的上传

在 8.4 节中，在创建公文文档的时候采用的是简单的文本编辑框，所有的内容都是由键盘输入的。在输入过程中，文档的格式还要通过添加一些 HTML 标签进行控制，这在实际使用中显然是不行的。为了弥补在线编辑的不足，很多公文往往是以附件的形式传送的，这也有利于用户的阅读与收藏。另外要说明的是，在实际项目开发过程中会充分利用现有的一些框架技术来增加系统的功能，例如 Zend Framework 支持良好的 Dojo，还有大家非常熟悉的 jQuery 等。由于本书主要介绍 Zend Framework 技术，所以没有用到这些框架，以使问题尽量简单化。

在 Zend Framework 项目的开发过程中，通常采用两种方法完成文件的上传操作：一种是采用 Zend_File 组件，就像第 7 章中实现用户信息的批量注入时那样；另一种就是直接采用 Zend_Form 的文件表单元素来完成文件的上传操作，下面分别进行介绍。

8.6.1 文件上传的表单方法

在第 6 章中介绍了 Zend_Form 的标准表单元素，并给出了它们对应的类。文件上传表单元素不在其标准表单元素之列，它所对应的类为 Zend_Form_Element_File，可以通过该类创建文件上传表单元素。在图 8.13 所示的表单中，文件上传表单元素是通过 Zend_Form 的 createElement 方法创建的。

打开 Document 控制器文件，在 issue 方法中添加代码：

```
public function issueAction()
{
        …
        if($this->getRequest()->isPost()){
            if($formFile->isValid($_POST)){
                //附件上传
                $path = APPLICATION_PATH.'/../public/uploads/'.
                                                date('Y-m-d').'/fileattach/';
    $folder = new Zend_Search_Lucene_Storage_Directory_Filesystem($path);
                $formattach = $formFile->getElement('attach');
                $formattach->setDestination($path);
                $fileinfo = pathinfo($formattach->getFileName());
                $extName = $fileinfo['extension'];
```

```
            $filename = 'attach'. $this->_user->getIdentity()->
                                        id.time().'.'. $extName;
            $formattach->addFilter('Rename',
                    array('target'=>$filename,'overwrite'=>true));
            if($formattach->receive()){
                $newinfo = pathinfo($formattach->getFileName());
                $file = date('Y-m-d').'/fileattach/'
                                        . $newinfo['basename'];
            }else {
                throw new Zend_Exception('文件附件上传失败!');
                exit();
            }
            $data = $formFile->getValues();
            //读入数据准备
            …
            $data['attach'] = $file;
            $newPage = $this->_page->addPage($data);
            $this->redirect('/document/docdetail/id/'. $newPage);
        }
    }
    $this->view->formFile = $formFile;
}
```

从代码中的变量 path 的值可以看出，上传的公文附件存放在系统目录下的 public\uploads\date('Y-m-d')\fileattach 文件夹中，其中的 date('Y-m-d')是用“年-月-日”构造的动态目录名，其效果如图 8.14 所示。

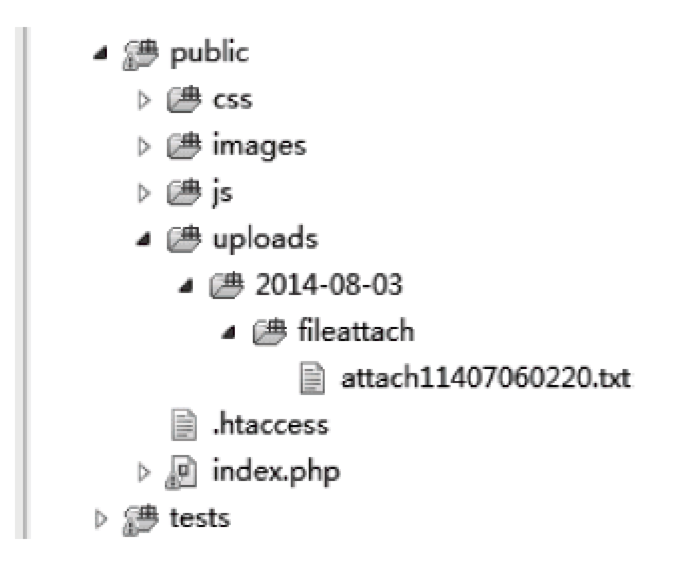


图 8.14　公文附件的存储目录结构

接着分析上述代码，确定了文件上传的目录后，接下来用参数 path 实例化一个 Zend_Search_Lucene_Storage_Directory_Filesystem 类对象。这个类实现了针对文件系统的目录功能，如果参数是一个字符串，则该类会认为这个字符串是一个文件系统的路径，若这个路径不存在，就会自动创建。

代码中的 formattach 变量就是文件表单中的“附件上传”表单元素，通过它调用 setDestination 方法设置 path 为文件上传的目标地址。变量 filename 为上传文件到服务器后的新文件名，这里是用 attach+“用户的 id”+“当前时间戳”+“原文件扩展名”来构成，这样可以根据文件名确定该文件的类型、上传的时间与上传操作者。

该代码中调用 addFilter 方法设置文件上传过滤器，主要功能是用新文件名替代原有文件的名字。一切都准备好后，调用 Zend_Form_Element_File 类的 receive 方法执行文件上传操作。如果没有异常，记录上传文件的目录及名字，以备写入数据库。

文件成功上传后，还需要在视图文件的适当位置输出其链接。打开 Document 控制器的 docdetail 视图文件 application\views\scripts\document\docdetail.phtml，添加代码：

```
…
<p><?php echo $this->docPage['body'];?></p>
<hr />
```

```
<?php if( $ this - > docPage[ 'attach']) :?>
< span >附件:     < a href = "<? php echo '/uploads/'. $ this - > docPage
[ 'attach'];?>">单击这里查看附件</a></span>
< hr />
<?php endif;?>
<?php
    …
?>
< hr />
```

公文附件上传成功并单击链接后的效果如图 8.15 所示。这里演示的是基本操作,在实际项目中还必须对上传文件的类型进行验证,不允许用户上传 exe、php 等危险文件到服务器。



图 8.15 公文上传附件的使用

8.6.2 文件上传的组件方法

文件上传的组件方法与上面介绍的 Zend_Form 表单方法基本相似,只是使用的适配器不同而已。下面的代码是在第 7 章的用户信息后台管理中使用过的,实现用户信息的批量注入。这里采用的是 Zend_File_Transfer_Adapter_Http 文件上传适配器。

```
public function insertAction()
{
      …
      if ( $ this - > getRequest() - > isPost()){
       $ adapter = new Zend_File_Transfer_Adapter_Http();
       $ path = APPLICATION_PATH. '/resources/'
                              .date('Y - m - d'). '/userInformation/';
    $ folder = new Zend_Search_Lucene_Storage_Directory_Filesystem( $ path);
        $ fileInfo = $ adapter - > getFileInfo();
```

```
        $extName = 'csv';
        $filename = 'admin'.$this->_user->getIdentity()->
                                              id.time().'.'.$extName;
            $adapter->addFilter('Rename',array(
            'target'=>$filename,'overwrite'=>true));
            $adapter->setDestination($path);
            $adapter->addValidator('Extension',false,array('csv'));
            if($adapter->receive()){
                $file = $path.$filename;
            }else{
                throw new Zend_Exception('文件上传失败!');
                exit();
            }

        $modeUser = new Admin_Model_User();
        if (($handle = fopen($file,'r')) !== FALSE) {
            $columns = fgetcsv($handle,1000,",");
        }
        while (($data = fgetcsv($handle,1000,",")) !== FALSE) {
            $info = array_combine($columns,$data);
            $addLastid = $modeUser->addByInfo($info);
            $addNum++;
        }
        fclose($handle);

         …
    }
```

8.7 本章小结

本章主要介绍系统公文信息管理功能的实现，包括数据库设计与业务逻辑两大部分。通过前面几章的学习，我们对 Zend Framework 框架的全貌已经有了比较清晰的了解，从本章开始，把重点转移到 Zend Framework 的组件技术上，即关注如何使用框架提供的组件高效、快捷地实现应用的业务逻辑。

本章介绍的 Zend Framework 新组件主要是 Zend_Paginator 与 Zend_File，它们分别完成数据的分页控制以及文件的上传与下载功能。这些都是 Web 应用中的常用技术，希望大家重点掌握。

第 9 章　留言信息管理模块

应用系统的内部留言，即个人消息的发送与接收，也是办公自动化管理系统中不可或缺的功能之一。它不同于 QQ 等即时通信工具，也不同于常用的邮件发送系统。使用即时通信工具时，要求连接外部网络，并且通信双方必须同时在线；使用邮件系统时，不仅要求连接外网，同时还需要用户具有确定的邮件接收地址。内部留言系统是应用系统内部用户之间短暂交流与沟通的平台，它其实不针对具体的用户，只是将全部信息保存于系统数据库中供用户查询而已。

本章仿照常用的 Web 邮件系统的外观实现系统的内部留言功能，由此介绍 Web 应用中的社区或论坛开发的业务逻辑。

9.1　留言功能预览

本系统的留言功能仿照常用的 Web 邮件系统实现，主要有个人消息的接收、管理与发送。其中，消息管理设置了收件箱、草稿箱、发件箱与垃圾箱功能，以便用户对个人消息进行分类管理。下面先来预览一下留言功能实现后的页面效果。

1. 系统主页个人消息显示效果

用户通过认证进入系统主页后，在系统主页的显示区有一个名为“最新消息”的版块，这里显示的是用户个人最新消息信息，如图 9.1 所示。



图 9.1　系统主页最新消息

2. 系统留言收件箱显示效果

图 9.2 所示为用户个人消息主页面效果，同时也是系统留言收件箱页面，这里显示用户收到的系统留言信息，主要是用红色图标标识的未读信息。



图 9.2　用户消息主页页面

3. 系统留言发件箱显示效果

图 9.3 所示为用户个人消息发件箱页面效果，这里显示用户已发出的留言信息。



图 9.3　用户信息发件箱页面

4. 系统留言发送消息页面效果

图 9.4 所示为用户个人消息编辑表单页面效果，表单中有接收消息人姓名、消息主题以及消息详情 3 项内容，接收信息人(好友)可以通过列表选取。



图 9.4 用户信息发送页面

9.2 数据库设计

为了实现用户个人消息的发送与分类显示，需要按照业务处理流程设计相应的数据库。本系统通过“发送消息”、“接收消息”和“用户好友”3 张数据表配合模型与视图来实现留言信息管理的功能。

用户个人留言消息从编辑到处理完毕大致要经过编辑、发送、接收和处理 4 个过程。用户在如图 9.4 所示的表单中编辑好消息内容后，可以单击“保存到草稿箱”按钮将消息进行保存，也就是说，消息暂不发送；也可以单击“发送消息”按钮，直接将消息发送出去。这个功能通过修改“发送消息”数据表中的“消息发送状态”字段值来完成。另外，消息的阅读与否、删除与否等操作也都可以采用同样的方法通过修改数据表中的相应字段值来实现。

1. 发送消息数据表

打开 phpMyAdmin 数据库管理工具，在 db_wmoams 数据库中新建名为 tb_send 的数据表，其字段含义见表 9.1，字段属性如图 9.5 所示。

表 9.1 tb_send 数据表的字段说明

字 段 名	说 明
id	自增变量，主键，消息在系统中的唯一标识
sendto	消息接收人
uid	消息发送人
title	消息标题

续表

字　段　名	说　　明
sendedflg	发送与否
delflg	删除与否
contents	消息内容
udat	更新时间
cdat	发送时间

名字	类型	排序规则	属性	空	默认	额外
id	int(11)			否	无	AUTO_INCREMENT
sendto	varchar(20)	utf8_general_ci		否	无	
uid	varchar(20)	utf8_general_ci		否	无	
title	varchar(400)	utf8_general_ci		否	无	
contents	text	utf8_general_ci		是	NULL	
sendedflg	int(1)			是	0	
delflg	int(1)			是	0	
cdat	datetime			否	无	
udat	datetime			是	NULL	
sdat	datetime			否	无	

图 9.5　tb_send 数据表的属性设置

注意：tb_send 数据表中既存储发送信息也存储回复信息。因为，对于某条信息的回复，相对于该消息的接收人来说也是一个消息发送的过程。

2. 接收消息数据表

使用 phpMyAdmin 数据库管理工具在 db_wmoams 数据库中新建名为 tb_receive 的数据表，其字段含义见表 9.2，字段属性如图 9.6 所示。

表 9.2　tb_receive 数据表字的段说明

字　段　名	说　　明
id	自增变量，主键，回复消息的唯一标识
sid	回复消息在 tb_send 中的 ID
uid	消息回复人
title	消息标题
delflg	删除与否
readflg	阅读与否
udat	更新时间
cdate	回复时间

名字	类型	排序规则	属性	空	默认	额外
id	int(11)			否	无	AUTO_INCREMENT
sid	int(11)			否	无	
delflg	int(1)			是	0	
readflg	int(1)			是	0	
udat	datetime			是	NULL	
cdate	datetime			是	NULL	

图 9.6　tb_receive 数据表的属性设置

3. 用户好友数据表

使用 phpMyAdmin 数据库管理工具在 db_wmoams 数据库中新建名为 tb_friend 的数据表，其字段含义见表 9.3，字段属性如图 9.7 所示。

表 9.3　tb_friend 数据表的字段说明

字　段　名	说　　明
id	自增变量，主键，好友表的唯一标识
uid	用户 ID
friend	用户好友表字符串
udat	更新时间
cdat	创建时间

名字	类型	排序规则	属性	空	默认	额外
id	int(10)			否	无	AUTO_INCREMENT
uid	int(4)			否	无	
friend	varchar(20)	utf8_general_ci		是	NULL	
cdat	datetime			是	NULL	
udat	datetime			是	NULL	

图 9.7　tb_friend 数据表的属性设置

注意： tb_friend 数据表的 friend 字段中存储的是由用户所有好友姓名组成的字符串，在使用时需要将其分离。

9.3　消息的接收

用户通过选择“个人办公”下的“个人消息”菜单项或单击系统主页左侧的快捷菜单“我的办公桌”中的“个人消息”链接都可以进入如图 9.2 所示的用户消息接收页面。用户的个人消息通过列表的形式在这里显示，其中，发件人旁边的红色信件图标表示该消息还未阅读；白色图标则表示消息已阅读。

9.3.1　创建控制器及方法

为了对系统留言信息进行管理，需设置相应的控制器及方法。控制器及方法的创建还是采用 Zend Framework 提供的工具来完成。

1. 创建控制器

打开 Zend Studio 集成开发环境，通过菜单打开 ZF Tool 工具窗口，输入命令：

```
zf create controller Message
```

创建用于个人消息管理的控制器 Message。打开 Message 控制器文件 application\controllers\MessageController.php，添加一些初始化代码，如下所示：

```
class MessageController extends Zend_Controller_Action
{
    protected $_receiveMess = null;
```

```
    protected $_sendMess = null;
    protected $_user = null;
    protected $_friend = null;
    public function init()
    {
        $auth = Zend_Registry::get('auth');
        if($auth->hasIdentity()){
            $this->_user = $auth;
        }
        $this->_receiveMess = new Wm_Model_Receive();
        $this->_sendMess = new Wm_Model_Send();
        $this->_friend = new Wm_Model_Friend();
    }
    public function indexAction()
    {
        $this->redirect('/message/receive');
    }
}
```

其中，对象 user 是已通过认证的登录用户；receiveMss、sendMess 以及 friend 分别为消息接收表 tb_receive、消息发送表 tb_send 和好友表 tb_friend 的表模型对象。

2. 创建控制器方法

选择 Zend Studio 集成开发环境工作区中的项目 wmProject，定位到项目的任意一个文件夹或文件。打开 ZF Tool 工具窗口，分别输入如下命令，给 Message 控制器添加 4 个方法。

```
zf create action receive Message
zf create action send Message
zf create action list Message
zf create action friendlist Message
```

9.3.2 创建表模型及方法

1. 创建表模型

在 Zend Studio 集成开发环境的 ZF Tool 工具窗口中分别输入如下命令，添加 9.3.1 节在控制器初始化函数中用到的 3 个表模型。

```
zf create model Receive
zf create model Send
zf create model Friend
```

分别打开 3 个模型文件，为其设置基类及对应的数据表名称，代码如下：

```
class Wm_Model_Receive extends Zend_Db_Table
{
    protected $_name = 'tb_receive';
}
```

```
class Wm_Model_Send extends Zend_Db_Table
{
    protected $_name = 'tb_send';
}

class Wm_Model_Friend extends Zend_Db_Table
{
    protected $_name = 'tb_friend';
}
```

上述模型类的基类也可以使用 Zend_Db_Table_Abstract 类，这个类其实也是 Zend_Db_Table 类的基类。

2. 添加表模型方法

表模型方法与具体的业务逻辑有关，下面先在 Wm_Model_Send 模型中添加一个 getMessages 方法，用它获取发送给用户的个人消息。代码如下：

```
public function getMessages($where = array(), $order = null)
{
        $select = $this->select()->setIntegrityCheck(false);
        $select->from('tb_send','*');
        if (count($where) > 0){
            foreach ($where as $key => $value){
                $select->where('tb_send.'.$key.' = ?', $value);
            }
        }
        if ($order){
            $select->order('tb_send'.$order);
        }else{
            $select->order('tb_send.sdat desc');
        }
        $select->join('tb_receive','tb_send.id = tb_receive.sid','readflg');
        $result = $this->fetchAll($select);
        if ($result){
            return $result;
        }
        else{
            return null;
        }
    }
```

该方法返回发送给某个用户的个人消息，其中的查询条件通过“键/值”数组的形式带入到方法中。结果集的排序按照用户的指定执行，如果用户没有指定，则按消息发送的时间降序排列。该方法采用了联合查询的方式，因为在显示收件箱中的消息时要区分该消息是否已经阅读过，而对“是否阅读”的判断是通过 tb_receive 表中的 readflg 字段的值来完成的，所以，在传给视图的查询结果中应该包括 readflg 字段的值。

9.3.3 设计视图文件

1. 添加控制器方法代码

准备好了数据库、表模型，就完成了业务逻辑的处理工作。为了将消息在页面上显示出

来，还必须有控制器方法与视图文件的配合。

打开上面创建的 Message 控制器文件 application\controllers\MessageController.php，在 receive 方法中添加如下代码：

```
public function receiveAction()
{
        $username = $this->_user->getIdentity()->username;
        $where = array('sendto' => $username, 'sendedflg' => 1);
        $messages = $this->_sendMess->getMessages($where);
        $this->view->messages = $messages;
}
```

在方法中，首先通过保存的用户认证信息获取用户姓名(这里用真实姓名进行查询)；然后构造查询条件，即查询写给用户 username 并且已经发送的消息；最后将查询条件带入模型方法，并将结果集传递给视图。

2. 编写视图文件代码

从本章 9.1 节展示的效果可以看出，个人消息显示页面主要由导航菜单区、功能操作区与列表显示区构成，为此，将视图文件也进行适当分割。页面的 CSS 样式文件统一存放在 messagecss.php 的文件中；左侧的导航菜单存放在 messageleft.phtml 中，在控制器方法对应的视图文件中只存放右侧的功能区与列表显示区的内容。

导航菜单视图文件 messageleft.phtml 的代码如下：

```
<div id="mssidebar">
    <div id="topmenu" align="left">
          <table border="0"><tr>
              <td id="lefttitle">个人消息</td>
          </tr></table>
    </div>
    <ul class="sidemenu">
            <li><a href="/message/receive">
                 <img src="/images/inbox.png" alt="收件箱">收件箱</a></li>
            <li><a href="#"><img src="/images/draft.png" alt="草稿箱">
                     草稿箱</a></li>
            <li><a href="/message/send">
                 <img src="/images/senti.png" alt="送件箱">发件箱</a></li>
            <li><a href="#"><img src="/images/delitem.png" alt="垃圾箱">
                     垃圾箱</a></li>
            <li><a href="#"><img src="/images/delitem1.png" alt="清空">
                     清空垃圾箱</a></li>
        </ul>
    </div>
```

视图文件 receive.phtml 的代码如下：

```
<?php require_once 'messagecss.php'?>
<?php require_once 'messageleft.phtml'?>
<div id="message"><div id="mscontent">
    <h2  class="pagetitle">系统内部留言：<b>收件箱 </b></h2>
    <div align="left">
```

```
            <ul class = "messagemenu">
                <li><a href = "#"><img src = "/images/addBtn.jpg"
                         alt = "新消息"></a></li>
                <li><a href = "#"><img src = "/images/queryBtn.jpg"
                         alt = "检索"></a></li>
            </ul>
            <hr />
        </div>
        <div id = "msmain">
            <form name = "listForm" action = "#" method = "POST">
                <input type = "hidden" name = "mode" value = "recieve">
                <input type = "hidden" name = "max" value = "1">
                <div align = "left" style = "font - size:0.8em">
                <a id = "allselect" href = "javascript:;" onclick = "setSelect()">
                       全部选择 </a> | <a id = "allrelease" href = "javascript:;"
                              onclick = "setRelease()">解除全部选择</a>  |
                              <a id = "allreverse" href = "javascript:;"
                            onclick = "setReverse()">全部反转</a>|</div>
    <br />
    <div  align = "left">
          <input class = "blog_delete" type = "button"
                     onclick = "listChk()" value = "删除">
    </div>
    <br/>
    <table class = "table - list"><thead><tr>
          <th class = "curve - left">发件人</th>
          <th>发送时间</th>
          <th class = "curve - right">标题</th>
    </tr></thead><tbody>
    <?php foreach ( $this - > messages as  $key = > $ message) :?>
    <tr class = "odd"><td width = "100" valign = "top" class = "line - left nowrap"
                     align = "left" rowspan = "1">
          <input type = "checkbox" name = "mid0" value = "11" />
                <img src = "/images/
    <?php echo  $ message['readflg'] ? 'obj_read.gif' : 'obj_yet.gif' ?>" >
              <?php echo  $ message['uid'] ?></td>
        <td class = "titlecell"><?php echo  $ message['sdat'] ?></td>
        <td rowspan = "1" align = "left" class = "line - right">
            <a href = "#"><?php echo  $ message['title'] ?></a></td>
        <input type = "hidden" name = "mode0" value = "recieve" />
    </tr><?php endforeach;?></tbody><tfoot><tr>
    <td class = "curve - left"></td><td></td><td class = "curve - right"></td>
    </tr></tfoot></table><br />
    <div align = "left" style = "font - size:0.8em">
    <a id = "allselect" href = "javascript:;" onclick = "setSelect()">全部选择 </a>
    | <a id = "allrelease" href = "javascript:;" onclick = "setRelease()">
    解除全部选择</a>  | <a id = "allreverse" href = "javascript:;"
    onclick = "setReverse()">全部反转</a>|   </div></form></div></div></div>
```

上述代码比较多，里面大部分都是关于页面布局的，在学习时请大家参考源代码，要特别注意代码中的阴影部分。接收个人消息的 M-V-C 现在都已经完成了，打开 Apache 服务

器，在浏览器的地址栏中输入 http://wmoams.com/message，效果如图 9.8 所示。



图 9.8　用户消息收件箱页面

9.4　消息的发送

上面实现了用户个人消息的接收，本节实现消息的发送。发送消息是通过用户单击如图 9.8 所示页面中的“新增”按钮来实现的，该按钮对应 Message 控制器中的 send 方法。

9.4.1　设计输入表单视图

消息发送时，用户需要在页面的输入表单中填写几项关键内容，这些内容主要有收件人姓名、消息主题、消息内容等。为了简便，这里使用 HTML 表单接受用户的输入，且只进行简单的有效性验证。当然，大家也可以用 Zend_Form 表单，与 HTML 表单相比，Zend_Form 表单具有强大的验证功能，但其装饰显得有些麻烦。

在 9.3 节中已经创建了名为 send 的 Message 控制器方法，现在打开这个方法所对应的视图文件 application\views\scripts\message\send.phtml，添加如下代码：

```
<?php require_once 'messagecss.php'?>
<?php require_once 'messageleft.phtml'?>
<div id="message"><div id="mscontent">
…
<form name='createform' action="/message/send" method="POST" onSubmit="return
checkSubmit()">
<table><tr>
    <td width="10%">收件人</td>
    <td style="width:70px;"><input type="text" id="user" size="20"></td>
    <td style="width:30px;"><input id="adduser" class="blog_delete" type="button"
onclick='checkUser("#");' value="添加到朋友表"/></td>
```

```
    <td><img id="list" src="/images/friend.gif" onclick=" displayList()" title="朋友
列表"></td>
    <td><input name="users" id="users" type="hidden" value="">  </td></tr>
<tr><td>主题词</td>
<td colspan="4"><input type="text" name="title" size="48" maxlength="120" value="">
</td>
</tr><tr><td colspan="5"><textarea name="contents" rows="9" cols="54"></textarea>
</td></tr><tr>
<td><input class="blog_delete" type="submit" onclick="this.form.mode.value='2'" value="保
存为草稿"></td>
<td colspan="4"><input class="blog_delete" type="submit" value="发送消息"></td></tr>
</table></form></div></div>
```

视图文件被渲染后的页面效果如图 9.9 所示。



图 9.9　用户消息添加页面

页面中除"收件人"、"主题词"和"消息内容"表单元素之外，还有 3 个按钮和一张图片。3 个按钮为"添加到朋友表"、"保存为草稿"和"发送消息"，它们分别完成将新收件人添加到用户好友列表、将待发消息保存到草稿箱和直接发送消息的功能。页面中的图片是用户好友列表链接，单击该图片会打开用户的好友列表，供用户从中选择收件人。

9.4.2　处理用户消息表单

1. 好友信息的显示

用户单击图 9.9 页面中的"好友"图片，会在页面中显示该用户的好友列表供用户选择，好友列表中的好友姓名从数据表 tb_friend 中获取。

1）添加视图代码

打开视图文件 application\views\scripts\message\send.phtml，在末尾添加如下代码：

```
<div id="FriendList" style="display:none;position:absolute;top:180px;
        left:900px;width:80px;height:200px;background:#CCC;float:left">
    <?php echo $this->action('friendlist','message')?>
</div>
```

上述代码中的阴影部分使用 Zend Framework 的视图助手调用了 Message 控制器的 friendlist 方法，这个方法暂时还没有业务逻辑代码，稍后再添加。

为了响应用户对“好友”图片的单击操作，在视图文件的前面添加如下 JavaScript 函数 displayList。代码中的函数 setText 是用户单击好友列表中的姓名后将值赋给“收件人”表单元素的函数。

```
<script type="text/javascript">
    function displayList(){
        var friendlist = document.getElementById("FriendList");
        var displayMode = friendlist.style.display;
       if(displayMode == 'none'){
          friendlist.style.display = "";
       }else{
          friendlist.style.display = "none";
       }
    }
    function setText(x){
        var userInput = document.getElementById("user");
        userInput.value = x;
   }
</script>
```

2）添加模型方法

打开模型文件 application\models\Friend.php，在 Wm_Model_Friend 类中添加如下方法：

```
class Wm_Model_Friend extends Zend_Db_Table
{
    protected $_name = 'tb_friend';
    public function getFriends($where = array())
    {
        $select = $this->select();
        if (count($where) > 0){
            foreach ($where as $key => $value){
                $select->where($key.' = ?', $value);
            }
        }
        $result = $this->fetchRow($select);
        if ($result){
            return $result;
        }
        else{
            return null;
        }
    }
}
```

3）添加控制器方法

打开上面创建的 Message 控制器文件 application\controllers\MessageController.php，在 friendlist 方法中添加如下代码：

```
public function friendlistAction()
{
        header('content-type:text/html; charset=utf-8');
        $uid = $this->_user->getIdentity()->id;
        $strFriends = $this->_friend->getFriends(
                                array('uid'=>$uid))->toArray();
        $friends = explode('#', $strFriends['friend']);
        $str = '';
        foreach ($friends as $k=> $key) {
            $str .= '<div style="width:95%; clear:both;
                    background-color:#FFFFFF; cursor:pointer;"
                    onmouseover="this.style.background=\'#FF8900\';
                    this.style.color=\'#FFFFFF\'"
                    onmouseout="this.style.backgroundColor=\'#FFFFFF\';
  this.style.color=\'#333333\'" onclick="setText(\''. $key . '\')">';
            $str .= '<ul>';
            $str .= '    <li style="font-size:13px;width:60%; height:18px;
 line-height:18px; text-align:center; float:left;">' . $key . '</li>';
            $str .= '</ul>';
            $str .= '</div>';
        }
        echo $str;
        $this->_helper->viewRenderer->setNoRender();
}
```

该方法通过用户 ID 调用 Wm_Model_Friend 模型中的方法获取登录用户的好友信息，并用列表的形式在视图页面中进行输出。

注意方法中的 4 个阴影代码块，第 1 个阴影代码块设置好友列表页面编码；第 2 个阴影代码块利用 PHP 的函数 explode 将好友列表字符串分离为单个的“姓名”，数据表 tb_friend 中的 friend 字段中存储的是由好友姓名组成的用“#”号分隔的字符串，见表 9.2；第 3 个阴影块循环输出好友姓名，并设置了用户单击列表项的操作函数 setText，该函数功能见上述说明；最后一句阴影代码使用 Zend Framework 的动作助手，关闭了控制器方法对应的视图，这里为 application\views\scripts\message\friendlist.phtml 视图，因为好友列表的显示需要在 send.phtml 中进行。关于 Zend Framework 的“助手”，将在第 10、11 章详细介绍。

通过上述 3 个步骤的准备之后，用户单击图 9.9 页面中的“好友”图片将会在页面中显示该用户的好友姓名列表，选择列表中的好友，在“收件人”表单元素中就会输入该好友的姓名，效果如图 9.10 所示。再次单击“好友”图片，关闭好友列表。

4）添加新好友

在上面的演示中，使用的示例数据都是直接在数据库中添加的。当用户在“收件人”表单元素中输入姓名后，可以单击页面中的“添加到朋友表”按钮将该收件人姓名添加到数据表 tb_friend 中。这个操作非常简单，请大家自己完成。需要提醒的是，在准备写入数据库

之前要对用户的输入进行有效性判断,并进行无害化处理,以防数据库攻击;在数据写入时还要进行是否重名的校核。



图 9.10　用户好友列表

2. 表单数据的处理

用户在页面中输入相关内容并通过验证后,如果单击"保存为草稿"按钮,则该消息被放入用户草稿箱中,暂缓发送。其实,此时只是将该消息保存到了 tb_send 数据表,并将字段 sendedflg 的值设置为"0"而已;但是,如果用户输入完毕,单击的是"发送消息"按钮,则完成数据的保存并设置字段 sendedflg 的值为 1 后,还需要将该消息放入 tb_receive 数据表中,以备检查消息的状态之用。

打开 Message 控制器文件,在方法 send 中添加代码:

```
public function sendAction()
{
        if($this->getRequest()->isPost()){
              $date = $this->getRequest()->getParams();
              $message['sendto'] = $date['sendto'];
              $message['title'] = $date['title'];
              $message['contents'] = $date['contents'];
              $message['uid'] = $this->_user->getIdentity()->username;
              $message['cdat'] = $message['udat'] =
                                   $message['sdat'] = date('Y-m-d H:i:s',time());
             if($date['mode'] == 1){
                   $message['sendedflg'] = 1;
                   $id = $this->_sendMess->addMessage($message);
                   $this->_receiveMess->addMessage(
                                   array('sid' => $id, 'udat' => $message['sdat']));
           }
           if($date['mode'] == 2){
               $message['sendedflg'] = 0;
               $this->_sendMess->addMessage($message);
           }
           $this->redirect('/message/list/type/send');
           }
}
```

该段代码分 3 个步骤完成消息的发送功能:首先判断方法的调用是否由用户单击了表单的"提交"按钮引起,若是,则取出提交的数据,并添加一些其他字段;然后判断用户单击

的“提交”按钮是“保存为草稿”还是“发送消息”；最后根据“提交”按钮的类型分别进行数据库操作。如果用户单击的是“发送消息”按钮，则将该消息同时写入 tb_send 和 tb_receive 数据表中；如果用户单击的是“保存为草稿”按钮，则只需要将数据写入 tb_send 数据表。

9.4.3 设计数据添加模型方法

上面在处理表单数据的 send 方法中用到了 tb_send 和 tb_receive 数据表模型的 addMessage 方法，这两个方法的实现逻辑是相似的，下面只编写 tb_send 表模型中的代码。

打开 application\models\Send.php 模型文件，添加如下代码：

```
public function addMessage( $data = array())
{
        $row = $this->createRow();
        if (count( $data) > 0){
            foreach ( $data as $key => $value){
                $row->$key = $value;
            }
            $row->save();
            return $row->id;
        }
        else{
            throw new Zend_Exception('向数据库中写入数据出错!');
        }
}
```

用户提交的数据以“键/值”数组的形式传入该方法，然后调用 Zend_Db_Table 表模型的 createRow 方法，创建一个 Zend_Db_Table_Row_Abstract 对象，也就是 tb_send 数据表的一个空“行”，接着依次读入在控制器的 send 方法中准备好的数据。

注意，数据数组的“键/值”一定要与表 tb_send 中的字段名称相同，同时还要注意表中字段的“空”属性设置，否则会抛出数据库写入异常。数据正确读入空“行”对象后，调用 Zend_Db_Table_Row_Abstract 类的 save 方法将数据写入数据库中。

9.5 消息的显示

9.5.1 消息的分类显示

从上面的页面效果图可以看出，系统的留言信息管理系统结构模拟 Web 邮件系统，将个人消息进行了分类存放。上面实现了查看“收件箱”中消息的功能，下面介绍“发件箱”、“草稿箱”和“垃圾箱”中消息的查询与显示。

1. 添加控制器方法代码

为了显示“发件箱”、“草稿箱”和“垃圾箱”中的消息，在 Message 控制器中添加一个名为 list 的方法。在 Zend Studio 集成开发环境中打开 Message 控制器文件，在 list 方法中添加代码：

```
public function listAction()
```

```
{
    $page = array();
    $type = $this->getRequest()->getParam('type');
    if($type == 'send'){
        $page['type'] = 'send';
        $page['title'] = '发件箱';
        $page['tbtitle'] = '收件人';
        $page['tmtitle'] = '发送时间';
        $uid = $this->_user->getIdentity()->username;
        $where = array('uid' => $uid, 'sendedflg' => 1, 'delflg' => 0);
        $messages = $this->_sendMess->getMessages($where);
    }
    if($type == 'draft'){
        $page['type'] = 'draft';
        $page['title'] = '草稿箱';
        $page['tbtitle'] = '收件人';
        $page['tmtitle'] = '更新时间';
        $uid = $this->_user->getIdentity()->username;
        $where = array('uid' => $uid, 'sendedflg' => 0, 'delflg' => 0);
        $messages = $this->_sendMess->getMessages($where);
    }
    if($type == 'rub'){
        $page['type'] = 'rub';
        $page['title'] = '垃圾箱';
        $page['tbtitle'] = '收件人/发件人';
        $page['tmtitle'] = '发送/更改时间';
        $uid = $this->_user->getIdentity()->username;
        $where1 = array('uid' => $uid, 'delflg' => 1);
        $messages1 = $this->_sendMess->getMessages($where1);
        $where2 = array('sendto' => $uid, 'delflg' => 1);
        $messages2 = $this->_sendMess->getMessages($where2);
        $messages = $messages1->toArray() + $messages2->toArray();
    }
    $this->view->page = $page;
    $this->view->messages = $messages;
}
```

这里所说的消息的分类实际上是虚拟的，数据库中并不存在与之对应的数据表，所以，在查看信息的时候必须知道它所在的虚拟分类名称。在上述 list 方法中，type 用来保存分类名称，它的值由视图页面中菜单的链接传递，其中字符串 send、draft、rub 分别表示“发件箱”、“草稿箱”和“垃圾箱”；代码中的 page 是自定义的页面信息数组，因为在显示不同的“箱”里的消息时，要显示的字段、标题文字是不一样的。例如，在查看发件箱时要输出“收件人”，而在查看垃圾箱时要输出“发件人”或“收件人”。

做好适当的准备工作之后，就可以分门别类地执行查询了。查询时直接调用模型中的 getMessages 方法就可以得到相应的结果集，这里的难点主要在查询条件的构造上。查询条件与数据库的设计有关，根据 9.2 节中给出的数据表结构，查询条件如下：

• 发件箱：由“我”创建、已经发送、没有被删除。即

```
array('uid' => $ uid, 'sendedflg' => 1, 'delflg' => 0)
```

- 草稿箱：由“我”创建、没有发送、没有被删除。即

```
array('uid' => $ uid, 'sendedflg' => 0, 'delflg' => 0)
```

- 垃圾箱：由“我”创建、已经被删除或发送给“我”、已经被删除。即

```
array('uid' => $ uid, 'delflg' => 1)
```

或

```
array('sendto' => $ uid, 'delflg' => 1)
```

2. 添加视图代码

list 方法的视图与 receive 方法的视图相似，只是要根据不同的类型进行输出。下面是部分代码：

```
…
<?php foreach ( $ this -> messages as $ key => $ message) :?>
<?php if( $ this -> page[ 'type'] == 'send') :?>
< tr class = "odd">
< td width = "100" valign = "top" class = "line - left nowrap" align = "left" rowspan = "1"> 
   < input type = "checkbox" name = " " value = "  " />< img src = "/images/obj_
read.gif" ><?php echo $ message[ 'sendto'] ?></td>
< td class = "titlecell"><?php echo $ message[ 'sdat'] ?></td>
< td rowspan = "1" align = "left" valign = "middle" class = "line - right">< a href = " /message/
detail/type/send/id/<?php echo $ message[ 'id'] ?>">
<?php echo $ message[ 'title'] ?></a></td>
</tr>
<?php endif;?>
…
```

9.5.2 消息的详细显示

1. 添加控制器方法

打开 Zend Studio 集成开发环境中的 ZF Tool 工具窗口，输入命令：

```
zf create action detail Message
```

在 Message 控制器中创建 detail 方法，并添加代码：

```
public function detailAction()
{
    $ page = array();
    $ type = $ this -> getRequest() -> getParam('type');
     $ id = $ this -> getRequest() -> getParam('id');
     $ message = $ this ->_sendMess -> find( $ id) -> current();
     $ page[ 'sendto'] = $ message[ 'sendto'];
     $ page[ 'sendtoname'] = '收件人';
     $ page[ 'sendby'] = $ message[ 'uid'];
     $ page[ 'sendbyname'] = '发件人';
```

```
    $page['messagetitle'] = $message['title'];
    $page['messagebody'] = $message['contents'];
    if($type == 'receive'){
        $page['time'] = $message['sdat'];
        $page['tmtitle'] = '发送时间';
    }
    if($type == 'send'){
        $page['time'] = $message['sdat'];
        $page['tmtitle'] = '发送时间';
    }
    if($type == 'draft'){
        $page['time'] = $message['udat'];
        $page['tmtitle'] = '更改时间';
    }
    if($type == 'rub'){
        $page['time'] = $message['udat'];
        $page['tmtitle'] = '更改时间';
    }
    $this->view->page = $page;
}
```

消息详细显示时，页面内容从数组 page 中取出，数组 page 中的值根据不同的消息来源在 detail 方法中赋值。上述代码中的第 2、3 行提取从视图页面传递过来的页面类型与消息的序号“id”，所以，必须在 receive. phtml 或 list. phtml 视图文件的链接中添加这两个参数。

2. 添加视图代码

打开 detail 方法的视图文件 application\views\scripts\message\detail. phtml，添加代码：

```
…
<div id="msmain">
<div align="left" style="font-size:0.8em"><a href="#" onclick="answer()">回信</a>
|<a href="#" onclick="forward()">转送</a>
| <a href="#" onclick="">删除</a>|<a href="#" onclick="editMessage()">
编辑此留言 </a>  |  <a href="#" onclick="">留言一览</a></div><br><hr />
<table>
<tr><td width="15%">消息标题：</td>
<td><?php echo $this->page['messagetitle']?></td></tr>
<tr><td><?php echo $this->page['sendbyname']?>:</td><td>
<?php echo $this->page['sendby']?></td></tr>
<tr><td><?php echo $this->page['tmtitle']?>: </td><td>
<?php echo $this->page['time']?></td></tr>
<tr><td><?php echo $this->page['sendtoname']?>: </td><td>
<?php echo $this->page['sendto']?></td></tr>
<tr><td>消息内容：</td><td colspan="4"><textarea name="contents" rows="3" cols="54">
<?php echo $this->page['messagebody']?></textarea></td></tr>
</table>
</div>
…
```

代码中省略的部分与 list. phtml 或 receive. phtml 相似，页面效果如图 9.11 所示。



图 9.11 用户消息详情页面

9.6 消息的移动与删除

上面为个人消息设置了虚拟的消息“箱”，实现了消息的分类管理。这一节继续完善消息的管理功能，实现消息的移动与删除。

9.6.1 消息的移动

所谓消息的移动，是指把消息从“收件箱”、“发件箱”、“草稿箱”转移到“垃圾箱”的过程以及其逆过程，下面用一个控制器方法来实现这个功能。

1. 添加控制器方法

打开 Zend Studio 集成开发环境，在 ZF Tool 工具窗口中输入命令：

```
zf create action move Message
```

在 Message 控制器中创建 move 方法，并添加代码：

```
public function moveAction()
{
    $type = $this->getRequest()->getParam('type');
    if($this->getRequest()->isPost()){
        $date = $_POST;
        if($type != 'rub'){
            foreach ($date as $mess){
                $this->_sendMess->updateMessage(
                                    $mess,array('delflg'=>1));
            }
            if ($type == 'receive') {
```

```
                $this->redirect('/message/receive');
            }else{
                $this->redirect('/message/list/type/'. $type);
            }
        }else{
            foreach ( $date as  $mess){
                $this->_sendMess->updateMessage(
                                        $mess,array('delflg'=>0));
            }
            $this->redirect('/message/list/type/rub');
        }
    }
    $this->redirect('/message');
}
```

程序首先判断方法的调用是否来自于表单的提交，注意这里的表单提交是指对页面中“删除”或“还原”按钮的操作。如果是用户单击了按钮，接着根据页面的类型进行不同的操作。如果消息不在“垃圾箱”中，则执行将消息移入“垃圾箱”的操作，否则，执行还原操作。

将消息移入“垃圾箱”的方法很简单，获取到选中消息的 id 后，调用 send 模型的 updateMessage 方法将该消息的 delflg 字段的值更改为“1”即可。updateMessage 方法是自定义的 send 模型方法，现在还不存在，我们在下面的小节中添加。消息的 id 是通过页面中的 checkbox 表单元素的值传递过来的。

把“垃圾箱”中的消息还原到原来的位置，只要将该消息的 delflg 字段的值重新还原为“0”即可。

2. 添加 send 模型方法

打开 send 模型文件 application\models\Send. php，添加 updateMessage 方法，代码如下：

```
public function updateMessage( $id, $data = array())
{
    $row = $this->find( $id)->current();
    if( $row){
        if (count( $data) > 0){
            foreach ( $data as  $key => $value){
                $row->$key =  $value;
            }
            $row->save();
            return  $row->id;
        }
        else{
            throw new Zend_Exception('向数据库中写入数据出错!');
        }
    }else {
        return false;
    }
}
```

上述代码非常简洁，只用了短短的 10 行就完成了数据库的更改操作，这就是使用面向

对象技术的优势所在。调用 updateMessage 模型方法只需要传入两个参数，一个是需要修改的消息的序号，用它可以准确地定位消息；另一个就是需要修改的字段与值组成的键/值数组。

代码中使用 Zend_Db_Table_Abstract 类的 find 方法，通过传入的消息序号获取到该消息的原始记录，然后对数据进行更改、保存。需要注意的是，find 方法是通过数据表的主键进行查找操作的，默认主键名为 id，如果数据表中的主键名不为 id，则需要在模型中进行声明。这个问题在前面的章节中已有说明，这里再强调一下。

3. 修改视图文件

模型和控制器准备就绪后，接下来修改视图文件。消息移动的效果是通过查询的方式展现出来的，在这里，我们弃用 move 方法的 move. phtml 视图，在 list 方法的视图文件 list. phtml 中修改代码完成消息的显示。下面只给出需要修改的关键代码。

```
<h2  class=" ">系统内部留言: <?php echo $this->page['title'] ?></h2>
```

从 list 方法传递给页面的 page 中取出消息“箱”的名称：

```
<form name="listForm"
        action="/message/move/type/<?php echo $this->page['type']?>"
method="POST">
```

将表单数据以 post 的方式提交到 Message 控制器的 move 方法进行处理，同时以 get 方式将页面类型 type 也传递到 move 方法中：

```
<input class="blog_delete" type="button" onclick="listChk()"
      value="<?php echo $this->page['type'] == 'rub' ? '还原':'删除'?>">
```

如果是在“垃圾箱”页面，将“删除”按钮的标签更改为“还原”：

```
<th class="curve-left"><?php echo $this->page['tbtitle'] ?></th>
<th><?php echo $this->page['tmtitle'] ?></th>
```

列表显示消息时，不同页面表头文字不一样：

```
<input type="checkbox" name="<?php echo 'mid'. $message['id'] ?>"
                          value="<?php echo $message['id'] ?>" />
```

这是消息列表中每条消息第一列中的多选表单元素（即 chechkbox）的 name、value 属性设置。这两处代码非常重要，用户的选择就是通过这种方式将消息的 id 传递到 move 方法中的。

另外，大家可能感到奇怪，在视图页面中并没有看到 form 表单的“提交”按钮，那么是怎样进行表单提交的呢？请查看页面中“删除”按钮的属性，里面有一个 onclick，它的值为 listChk()，这是 JavaScript 函数，代码如下：

```
function listChk(e){
  var inputList = document.listForm.elements;
  var count = 0;
  for (var i = 0; i < inputList.length; i++) {
     var input = inputList[i];
```

```
        if(input.type == 'checkbox' && input.checked == true)
            count++;
    }
    if(count == 0){
        alert('请选择要移动的消息!');
        return false;
    }
    document.listForm.submit();
    return true;
}
```

该函数先检查用户是否选择了需要移动的消息,如果有选择,则执行表单的提交操作。

到现在为止,M-V-C 模式中的 3 项都已准备完毕,下面就可以在浏览器中进行测试了,效果如图 9.12～图 9.14 所示。图 9.12 为"发件箱"页面,在这里选择了第 2 条消息,单击"删除"按钮,数据被提交到 move 方法,处理完毕后重新回到"发件箱",如图 9.13 所示。从该图中可以看出,move 方法正确移除了第 2 条消息。打开"垃圾箱",如图 9.14 所示,可以看到"发件箱"中的消息被移到"垃圾箱"中。



图 9.12　选择发件箱消息

9.6.2　消息的删除

消息的物理删除通过菜单中的"清空垃圾箱"完成。从上面的分析可知,"垃圾箱"中的消息既可能来自于 tb_send 表,也可能来自于 tb_receive 表。所以,删除消息必须将两个表中的关联数据全部删除,这种在关联表之间进行的操作称为级联操作。

级联操作可以在数据库层面解决,主流的数据库都支持 CASCADE 级联,所以通常涉及关联删除或更新时可以在数据库中使用 SQL 语句建立级联操作。Zend Framework 的数据库抽象层与数据库无关,如果更换数据库,Zend Framework 程序仍旧能正常工作,不会因此出现任何问题。所以,Zend Framework 的 Zend_Db 提供了代码层面的级联功能,以使



图 9.13　消息移动后的发件箱页面



图 9.14　垃圾箱页面效果

不依赖数据库就可以实现数据的级联操作。

1. 修改模型文件

如果要实现数据表 tb_send 与 tb_receive 的级联操作，需要对它们对应的模型文件进行修改，即在模型中定义级联关系。

打开 Send.php、Receive.php 模型文件，分别添加如下代码：

```
class Wm_Model_Send extends Zend_Db_Table
{
    protected $_name = 'tb_send';
```

```
    protected $_dependentTables = array('Wm_Model_Receive');
    …
}

class Wm_Model_Receive extends Zend_Db_Table
{
    protected $_name = 'tb_receive';
    protected $_referenceMap = array(
        'Post' => array(
            'columns'      =>  array('sid'),
            'refTableClass'  => 'Wm_Model_Send',
            'refColumns'     => array('id'),
            'onDelete'       => self::CASCADE
        )
    );
…
}
```

Zend_Db_Table_Abstract 通过_dependentTables 和_referenceMap 表示表间关系。dependentTables 表示依赖于本表的类名，它是一个数组，在其中可以配置多个依赖表。注意，这里使用的是表模型的类名，例如代码中的 Wm_Model_Receive。

如果要让 Zend_Db_Table 级联更新和删除，需要明确父表和子表，被依赖的表一般作为父表，依赖其他表的表作为子表，这里 tb_send 为父表、tb_receive 为子表。

在父表中配置了 dependentTables，在子表中也要相应配置 referenceMap。referenceMap 表示本表对其他表的依赖关系，它也是一个数组，数组的每一个元素代表一种映射关系，该映射关系由映射名、本表引用字段、被引用表名、被引用字段等构成，如代码所示。各配置项的含义如下：

- columns：用一个字符串或者数组表示的依赖表的外键。一般情况下，它是一个单独的字段，有些表也可能是多个字段。
- refTableClass：父表的类名，而不是表名。
- refColumns：父表中的主键，一般情况下它也是一个字段，有的表可能有多个字段。
- onDelete：父表中删除记录时的操作。
- onUpdate：父表中主键被修改时的操作。

模型经过如此配置后就具备了级联删除功能，所以当用户删除“垃圾箱”中的消息时，如果该消息来自于“收件箱”，则该消息在 tb_receive 中的记录也会一起被删除。

另外，在 tb_send 表模型中再添加一个删除方法，代码如下：

```
public function deleteMessage($id)
  {
      $row = $this->find($id)->current();
      if($row){
          $row->delete();
      }else{
          throw new Zend_Exception('删除消息出错!');
      }
  }
```

2. 创建控制器方法

打开 Zend Studio 集成开发环境的 ZF Tool 工具窗口，输入命令：

```
zf create action delete Message
```

在 Message 控制器中创建 delete 方法，并添加代码：

```
public function deleteAction()
{
        $ messages = $ this->_sendMess->getMessages(array('delflg'=>1));
        foreach ( $ messages as $ mess){
            $ this->_sendMess->deleteMessage( $ mess['id']);
        }
        $ this->redirect('/message/list/type/rub');
}
```

该方法首先获取 tb_send 中字段 delflg 的值为“1”的所有消息，这些消息就是“垃圾箱”中的全部消息；然后遍历结果集，提取消息的 id 并删除。由于已经定义了级联操作，所以删除操作虽然调用的是 tb_send 的表模型方法 deleteMessage，数据表 tb_receive 中的关联消息会一起被删除。

9.7 本章小结

本章详细介绍了系统内部留言功能的实现过程，是学习 Zend Framework 应用的工作原理、MVC 机制、数据库操作、layout 布局模板以及组件使用的又一综合实例。内部留言系统是应用系统内部用户之间短暂交流与沟通的平台，它综合了常用即时通信工具及邮件系统的特点，是办公自动化管理系统中不可或缺的功能模块。

本章留言系统界面是按照 Web 邮件系统的要求做成的，设置了收件箱、草稿箱、发件箱与垃圾箱以及消息发送页面。在学习过程中，在读者进一步巩固前面所学知识的同时应重点掌握利用数据库实现复杂业务逻辑的方法以及 Zend Framework 的数据库级联操作。

第 10 章　事务信息管理模块

行政事务是指国家机关、企事业单位和社会团体内部的日常管理事务与各项服务，例如收发文档、接待来宾、收支买办、车辆、安全、福利、卫生、后勤补给及保障等。这些事务虽然都比较琐碎，但至关重要，事务处理的高效与否直接关系到企业的内、外部形象。

本章实现办公自动化管理系统的部分事务管理功能，包括申请管理、报修管理、车辆管理等，由此介绍 Zend Framework 的视图助手，并进一步熟悉 Zend Framework 相关组件的使用。

10.1　事务信息管理效果预览

根据第 3 章的总体规划，本系统设置了专门的事务信息管理模块用来处理各部门及用户个人的行政管理事务。该模块功能位于导航菜单的“事务管理”主菜单下，包括“领导日程”、“公用申请”、“请假管理”、“印章管理”、“车辆管理”、“报告意见”等。这里的“领导日程”是指领导接待日程安排；“公用申请”是指一些临时性的安排申请，例如借用教室、会议室、报告厅、实验室等。由于事务管理非常琐碎，涉及的部门及人员也非常多，其实现比较复杂，这里只介绍“申请管理”功能的实现过程。下面是系统事务信息管理的部分页面效果。

1. 事务管理模块主页

图 10.1 所示为用户单击“事务管理”主菜单后的页面效果，页面左侧的快捷菜单设置了“待办事务”及“在办事务”两个子菜单项，方便用户随时查看事务完成的情况；页面右侧为重要事务及个人工作安排中急需完成的任务的“待办提醒”，它的内容由用户自行设置；中间部分为“事务管理”的主显示区。

2. 全部事务管理页面

图 10.2 所示为用户单击“事务管理”主菜单后的主页面效果，页面分左、右两部分，左侧为导航菜单，同时也是事务信息的分类显示菜单，这里设置了“全部事务”、“待办事务”、“在办事务”、“已办事务”、“在审事务”5 种分类形式；右侧又分为上、下两个区域，上面为操作区，下面为事务信息详细显示区。

该图中显示的是与该用户有关的全部事务，每件事务除显示开始日期、主题及内容外，还用红色字体显示了该事务的处理状态，让用户对已完成、未完成、审核中的事务都能一目了然。

3. 在审事务管理页面

图 10.3 所示为用户单击“在审事务”子菜单后的页面效果，页面右下区显示的是申请已经提交、正在等待批复的事务详情；上面操作功能区列出了登录用户能够对事务信息进行



图 10.1 事务管理主页面



图 10.2 全部事务页面

的各种操作，这些操作是由用户的权限决定的。

图中功能区显示了“批复”按钮，说明登录用户具有审批的权限，单击该按钮，进入“申请审批”页面，在这里可以查询到需要该用户审批的所有事务申请详情，并对每件事务进行处理。

4. 待办事务管理页面

图 10.4 所示为用户单击“待办事务”子菜单后的页面效果，页面右下区显示的是申请已经批复、正在等待用户处理的事务详情，包括事务申请原文及审核批复原文，同时在事务申请原文的标题旁设置“收件”超级链接，用于修改该事务的处理状态，用户单击该链接，即可将该事务状态从“待办”修改为“在办”。



图 10.3　在审事务页面

图 10.4　待办事务页面

5. 添加事务管理页面

图 10.5 所示为用户单击操作功能区中的“新增”按钮后的页面效果。添加事务信息就是向数据库表中写入数据，因此要求表单元素与数据表字段吻合。这里设置了“标题”、“类型”、“内容”及“审批人”4 种表单元素接收用户的输入。



图 10.5　事务添加页面

10.2 数据库设计

根据第 3 章的总体设计，本系统中事务信息管理的数据表由“事务表”与“批复表”组成。“事务表”用来记录全部的事务信息，对该表执行不同条件的查询，可以获取不同类型的事务信息的集合；“批复表”用来记录对事务申请的批复，它是“事务表”的子表。

10.2.1 事务信息数据表的设计

事务信息数据表存储用户提交的全部事务信息，为了方便查询与分类，通过数据表的字段能够获取事务的如下信息。

1. 事务类型

事务申请从本质上来说是前面第 8 章所述的“公文”的一种，因此它的类型应该是 tb_doctype 数据表中的值，它依赖数据表 tb_doctype。

2. 事务申请人

由于是 Web 应用，系统所有用户提交的事务申请均存储在该表中，因此必须记录事务的申请人信息。在这里可以直接用字符串记录申请人姓名，也可以先记录用户 id，再通过 id 在用户表 tb_user 中查询申请人的其他信息。如果采用后一种方法，则“事务表”又会依赖“用户表”tb_user。

3. 事务主题与内容

事务的主题与内容是事务的主体，通过数据库的模糊查询可以从事务的主题与内容详情中提取相关的分类信息。

4. 事务审核人

在添加事务申请的时候，用户必须选择该申请提交的部门或个人，不同类型的事务由不同的部门进行处理。事务的审核部门或个人也应该与“部门表”tb_depart 或“用户表”tb_user 相对应。

5. 事务处理状态

从一件事务的生命周期来看，它应该处于编辑、审核、待办、在办和办结中的某一个状态，不同状态的事务，其处理与显示方式的侧重点稍有不同。例如处于“审核”状态的事务，不仅申请人能够查询，同时必须让审核人也能够查询；而处于“待办”、“在办”等状态的事务，与审核人就没有太大的关系了。

这里所列举的只是一些常用的字段设置，在项目开发过程中还要根据具体情况进行适当的调整。表 10.1 列出了本系统事务数据表的字段名称及作用。

表 10.1 tb_affair 数据表的功能说明

字段名	说明
id	自增变量，主键，事务的唯一标识
typeid	事务类型，例如请示、报告、意见等，与 tb_doctype 数据表的 id 字段对应
title	事务主题
content	事务详情

续表

字段名	说　明
uid	事务申请人 ID,与 tb_user 数据表的 id 字段对应
cdat	事务申请提交时间
checkid	事务审核人 ID,与 tb_user 数据表的 id 字段对应
replyid	事务批复 ID,与 tb_reply 数据表的 id 字段对应
state	事务状态,0、1、2、3、4 分别表示编辑、审核、待办、在办和办结状态
prompt	事务摘要,用于"待办提醒"功能的显示

根据表 10.1 中的字段及功能说明,用命令方式或用 phpMyAdmin 数据库管理工具创建事务表 tb_affair,如图 10.6 所示。

名字	类型	排序规则	属性	空	默认	额外
id	int(4)			否	无	AUTO_INCREMENT
typeid	int(4)			否	无	
title	varchar(20)	gb2312_chinese_ci		否	无	
content	varchar(200)	gb2312_chinese_ci		否	无	
uid	int(4)			否	无	
cdat	int(11)			否	无	
checkid	int(4)			否	无	
replyid	int(4)			是	NULL	
state	tinyint(1)			否	无	
prompt	varchar(20)	gb2312_chinese_ci		否	无	

图 10.6　tb_affair 事务信息数据表

10.2.2　事务批复数据表的设计

事务批复数据表是事务信息数据表的子表,只需记录批复详情及审核人、批复日期即可。如果需要直接从"批复表"中查找该批复对应的事务申请,也可以设置一个记录事务 ID 的字段,让该字段与 tb_affair 的 id 相对应。

事务批复数据表的字段设置与作用见表 10.2。

表 10.2　tb_reply 数据表的功能说明

字段名	说　明
id	自增变量,主键,事务批复的唯一标识
title	事务批复主题
content	事务批复详情
checker	事务审核人 ID,与 tb_user 数据表的 id 字段对应
cdat	事务批复时间

根据表 10.2 中的字段及功能说明,用命令方式或用 phpMyAdmin 数据库管理工具创建事务表 tb_reply,如图 10.7 所示。

名字	类型	排序规则	属性	空	默认	额外
id	int(4)			否	无	AUTO_INCREMENT
title	varchar(20)	gb2312_chinese_ci		否	无	
content	varchar(200)	gb2312_chinese_ci		否	无	
checker	varchar(20)	gb2312_chinese_ci		否	无	
cdat	int(11)			否	无	

图 10.7　tb_reply 事务批复数据表

10.3　事务信息的显示

事务信息的显示过程实际上就是一个数据库的分类查询过程。从 10.1 节中给出的页面效果可以看出，本系统除了能显示全部事务的详情及状态外，还将事务分成了在审、待办、在办和已办 4 种类型。下面实现这些事务的分类显示。

10.3.1　创建控制器及方法

为了对事务信息进行管理，需设置相应的控制器及方法，控制器及方法的创建这里还是采用 Zend Studio 集成开发环境提供的 Zend Framework 工具完成。

1. 生成控制器及方法

启动 Zend Studio 集成开发环境，选择项目工作区中的 wmProject，通过菜单打开 ZF Tool 工具窗口，输入命令：

```
zf create controller Affair
```

在默认模块 default 中创建事务管理控制器 Affair。然后用同样的方法打开 Zend Tool 工具窗口，输入命令：

```
zf create action list Affair
```

在控制器 Affair 中创建 list 方法。

2. 初始化控制器

通过上面的 Zend Framework 工具创建了 Affair 控制器及方法 list，同时关联地创建了控制器的 init、index 方法以及相应的视图文件 index.phtml 和 list.phtml。为了下面编程方便，在 Affair 控制器中定义 user、affair、reply 3 个对象，分别表示登录用户、事务模型、批复模型，并在 init 方法中对其初始化。代码如下：

```
class AffairController extends Zend_Controller_Action
{
    protected $_user = null;
    protected $_affair = null;
    protected $_reply = null;
    public function init()
    {
        $auth = Zend_Registry::get('auth');
        if($auth->hasIdentity()){
```

```
            $this->_user = $auth;
        }
        $this->_affair = new Wm_Model_Affair();
        $this->_reply = new Wm_Model_Reply();
    }
...
}
```

代码中的 Wm_Model_Affair 与 Wm_Model_Reply 为 10.2 节中创建的 tb_affair、tb_reply 数据表的表模型，现在它们还不存在，将在 10.3.2 节中创建。用户信息从用户认证信息对象 auth 中提取。

10.3.2 创建数据表模型及方法

在上面的控制器初始化代码中创建了“事务表”模型对象 $_affair 与“批复表”模型对象 $_reply，通过这两个对象可以对“事务表”和“批复表”进行操作。

1. 创建模型

打开 Zend Studio 集成开发环境中的 ZF Tool 工具窗口，分别输入命令：

```
zf create model Affair
```

和

```
zf create model Reply
```

创建 Wm_Model_Affair 与 Wm_Model_Reply 模型类，为类添加基类及相应的数据表名，代码如下：

```
class Wm_Model_Affair extends Zend_Db_Table
{
  protected $_name = 'tb_affair';
...
}

class Wm_Model_Reply extends Zend_Db_Table
{
  protected $_name = 'tb_reply';
...
}
```

2. 添加方法

打开模型文件 application\models\Affair.php，为表模型 Wm_Model_Affair 添加 getAffairs 方法及 addAffair 方法，代码如下：

```
public function getAffairs($where = array(), $order = null)
{
    $select = $this->select();
    if (count($where) > 0){
        foreach ($where as $key => $value){
            $select->where($key.' = ?', $value);
```

```
            }
        }
        if ( $ order){
             $ select -> order( $ order);
        }
         $ result = $ this -> fetchAll( $ select);
        if ( $ result){
            return $ result;
        }
        else{
            return null;
        }
    }
```

表模型 Wm_Model_Affair 的 getAffairs 方法用来执行对数据表 tb_affair 的查询操作，查询条件由 where 从调用程序中带入，查询结果集的排序由变量 order 确定。这里的查询条件只考虑了键/值数组的形式，也就是说，查询的条件只能用“=”条件运算符进行处理，如果要在查询条件中使用“!=”、“>”、“<”等其他条件运算符，还必须在方法中添加能够处理以字符串形式表示的条件语句的代码。另外，方法中的查询与排序子句都是通过 where 与 order 方法的形式添加到查询对象中的。代码中的 select 为查询对象，它是 Zend_Db_Table_Select 实例，以这种方式进行查询，可以有效地防止数据库的注入攻击。

```
    public function addAffair( $ data = array())
    {
         $ row = $ this -> createRow();
        if (count( $ data) > 0){
            foreach ( $ data as  $ key => $ value){
                 $ row -> $ key = $ value;
            }
             $ row -> save();
            return $ row -> id;
        }
        else{
            throw new Zend_Exception('向数据库中写入数据出错!');
        }
    }
```

表模型 Wm_Model_Affair 的 addAffair 方法用来执行对数据表 tb_affair 的添加记录操作，记录数据由 data 数组带入。

在方法代码中，首先创建一个 Zend_Db_Table_Row_Abstract 对象，暂且把它理解为数据表 tb_affair 的一个空“行”，然后通过循环将带入的数据按照键/值对形式读入到这个空“行”中，最后保存并返回添加的新记录的 id 值。从这段代码可以看出，带入数据的 data 数组的“键”必须与数据表 tb_affair 的字段名完全吻合，且不能有多余的数据项，否则操作会抛出异常，请大家特别注意。

完成 Wm_Model_Affair 表模型的设计后，接着打开模型文件 application\models\Reply.php，为表模型 Wm_Model_Reply 添加 getReply 方法，代码如下：

```
public function getReply( $ where = null, $ order = null){
    if(is_numeric( $ where)){
        $ row = $ this -> find( $ where) -> current();
    }else{
        $ row = $ this -> fetchRow( $ where, $ order);
    }
    if ( $ row) {
        return $ row;
    }else {
        return null;
    }
}
```

10.3.3 事务信息的全部显示

图 10.2 所示为事务信息全部显示时的页面效果，它是在用户单击"事务管理"主菜单后出现的，显示在 Affair 控制器的 index 视图页面中。这里的全部事务信息是指用户提交的信息，不包括具有审批权限的用户的批复信息。批复信息在用户单击"批复"按钮后的视图页面显示。

1. 编写 index 方法代码

打开 Affair 控制器文件 application\controllers\AffairController. php，在其 index 方法中添加如下代码：

```
public function indexAction()
{
    $ uid = $ this -> _user -> getIdentity() -> id;
    $ affairs = $ this -> _affair -> getAffairs(array( 'uid' => $ uid), 'cdat desc');
    $ this -> view -> affairs = $ affairs;
}
```

要显示用户的所有事务信息，就是要在 tb_affair 数据表中查询字段 uid 的值等于用户 ID 的所有记录，因此首先必须获取登录用户的 ID。

代码中的第 1 句，通过前面定义的_user 对象调用 Zend_Auth 类的 getIdentity 方法，从登录认证信息中直接提取用户的 ID；第 2 句，通过_affair 模型对象调用 getAffairs 模型方法，带入 uid= $ uid 查询条件，获取所有用户事务信息，并对结果集按 cdat 字段降序排列；第 3 句，将查询到的结果集通过 affairs 传递到视图中。

2. 编写 index 方法的视图代码

根据 10.1 节中展示的页面结构，我们将事务信息管理模块的视图文件分成 3 个部分，即 affairmenu. phtml、index. phtm 和 affaircss. php，它们分别代表事务信息导航区、事务信息详情区与页面 CSS 布局。

index. phtm 视图文件的代码如下：

```
<?php require_once 'affaircss. php'?>
<?php require_once 'affairmenu. phtml'?>
< div id = "message">< div id = "mscontent">
< h2   class = "pagetitle">事务管理：< b>全部事务</b></h2 >
```

```
<div align="left">
    <ul class="messagemenu">
        <li><a href="/affair/apply">
                <img src="/images/addBtn.jpg" alt="添加"></a></li>
          <li><a href="#">
                <img src="/images/queryBtn.jpg" alt="检索"></a></li>
      </ul>
      <hr />
  </div>
  <div id="msmain">
  <?php foreach ($this->affairs as $affair):?>
  <?php
      switch ($affair['state']) {
          case 1:$str='[审核中...]';break;
          case 2:$str='[等待办理...]';break;
          case 3:$str='[正在办理...]';break;
          case 4:$str='[已经完成]';break;
          default:$str='';break;
      }
  ?>
  <table style="background:url(/images/date-bg2.gif)
                                 no-repeat left top;margin-top:5px">
      <tr><td width="60px" align="center">
              <?php echo date('m',$affair['cdat'])?>月</td>
      <td style="line-height:30px;border-bottom: 1px dashed #ff8800">
      "标题"<?php echo $affair['title']?>  <font color='red'>
              <?php echo $str ?></font></td></tr>
      <tr style="line-height:30px;border-bottom: 1px solid #ff8800">
        <td style="font-weight:bold;font-size:24px;text-align:center;">
              <?php echo date('d',$affair['cdat'])?></td>
        <td>「内容」<?php echo $affair['content']?></td></tr>
</table>
<?php endforeach;?>
</div>
</div></div>
```

代码中使用了 switch…case 结构来确定每件事务的处理状态文本,这是因为数据表 tb_affair 中的状态字段 state 的数据类型为整型,显示时要输出相应的文本。

在视图文件中使用这么多的 PHP 代码,在以 MVC 为模型的 Zend Framework 项目中显然不是一个好的处理方法,有悖于数据的处理与显示分离的程序设计原则。Zend Framework 提供了一种称为"视图助手"的处理技术来解决在视图文件中执行复杂操作的问题,相关内容将在稍后的小节中进行介绍。

接着看上述代码,代码的第 2 句包含了 affairmenu.phtml 视图文件,其内容如下:

```
<div id="mssidebar">
  <div id="topmenu" align="left"><table border="0" >
  <tr><td id="lefttitle">事务管理</td></tr></table></div>
          <ul class="sidemenu">
            <li><a href="/affair">
```

```
                    <img src="/images/calendar2.gif" alt="全部事务">
                            全部事务</a></li>
                <li><a href="/affair/list/state/2">
                    <img src="/images/clock.gif" alt="待办事务">
                            待办事务</a></li>
                <li><a href="/affair/list/state/3">
                    <img src="/images/info_query.gif" alt="在办事务">
                            在办事务</a></li>
                <li><a href="/affair/list/state/4">
                    <img src="/images/contract.gif" alt="已办事务">
                            已办事务</a></li>
                <li><a href="/affair/list/state/1">
                    <img src="/images/senti.png" alt="在审事务">
                            在审事务</a></li>
            </ul>
</div>
```

10.3.4 待办事务的显示

待办事务信息的显示效果如图 10.4 所示，页面中不仅显示了事务申请的详情，同时还显示了该事务的批复内容，以方便用户确定下一步的操作方式。上面介绍的所有事务的显示是由控制器的 index 方法实现的，属于事务信息管理模块的主页；“待办事务”、“在办事务”等分类显示用控制器的 list 方法统一完成。

1. 编写 list 方法代码

打开 Affair 控制器文件 application\controllers\AffairController.php，在其 list 方法中添加如下代码：

```
public function listAction()
{
    $page = array();
    $state = $this->getRequest()->getParam('state');
    $uid = $this->_user->getIdentity()->id;
    $role =  $this->_user->getIdentity()->role;
    switch ($state) {
        case '1':
            $page = array('title' =>'您的申请正在审核中,请等待...');
            $affairs = $this->_affair->getAffairs(
                    array('uid' => $uid, 'state' => 1), 'cdat desc');
        break;
        case '2':
            $page = array('title' =>'待办事务');
            $affairs = $this->_affair->getAffairs(
                    array('uid' => $uid, 'state' => 2), 'cdat desc');
        break;
        case '3':
            $page = array('title' =>'在办事务');
            $affairs = $this->_affair->getAffairs(
                    array('uid' => $uid, 'state' => 3), 'cdat desc');
        break;
```

```
                case '4':
                    $ page = array('title' = >'已办事务');
                    $ affairs = $ this - >_affair - > getAffairs(
                            array('uid' = > $ uid, 'state' = > 4), 'cdat desc');
                break;
            }
            $ page = array_merge(array('role' = > $ role), $ page);
            $ this - > view - > page = $ page;
            $ this - > view - > affairs = $ affairs;
        }
```

该方法通过一个 switch…case 结构，根据请求中的 state 参数值来确定事务信息的显示类型。在每个 case 小节中，程序均完成两件事情，一是设置视图页面的信息，即变量 page 的赋值；二是从数据表 tb_affair 中查询满足条件的事务信息集合，即为变量 affairs 赋值。查询过程调用上面定义的模型方法完成，代码非常简单，这里不再赘述。

代码中的变量 role 表示登录用户的"角色"，也就是用户权限，它的值从登录认证信息中提取得到。之所以要获取用户的权限，是因为在视图中的某些操作，比如"批复"等，是由不同的用户权限决定的，不同的用户打开的视图页面，其页面元素会有所不同。代码中的倒数第 4 行，使用函数 array_merge 将"用户权限"添加到 page 数组中，以便将其传递到视图中。

2. 编写 list 方法的视图代码

list 方法的视图代码与前述 index 方法的代码基本相似，不同的是在显示"待办事务"时需要同时显示批复的内容，因此在 list 方法的视图文件中需根据事务状态确定是否有批复内容显示。

list. phtm 视图文件的部分代码如下，省略的部分与 index. phtml 相同。

```
…
<?php foreach ( $ this - > affairs as $ affair) :?>
  < table style = "background:url(/images/date - bg2. gif)
                                    no - repeat left top;margin - top:5px">
      < tr >< td width = "60px" align = "center">
                  <?php echo date('m', $ affair['cdat'])?>月</td >
    < td style = "line - height:30px;border - bottom: 1px dashed # ff8800">
                  "标题"<?php echo $ affair['title']?>   
                  < a href = " # ">< font color = "red">
                        <?php echo $ affair['state'] == 2?'[收件]':''?>
                  </font ></a ></td ></tr >
          …
    <?php if( $ affair['replyid'] && $ affair['state'] == 2){
              $ model = new Wm_Model_Reply();
              $ reply = $ model - > getReply( $ affair['replyid']);
    ?>
    < table style = "background:url(/images/date - bg2. gif)
                              no - repeat left top;margin - top:5px">
        < tr >< td width = "60px" align = "center">
                  <?php echo date('m', $ reply['cdat'])?>月</td >< td style = "line - height:30px;
border - bottom: 1px dashed # ff8800">
```

```
                <font color = "red">"批复"</font>
                <?php echo $affair['title']?></td></tr>
            <tr style = "line-height:30px;border-bottom: 1px solid #ff8800">
          <td style = "font-weight:bold;font-size:24px;text-align:center;">
            <?php echo date('d', $reply['cdat'])?></td><td>
            「内容」<?php echo $reply['content']?></td></tr>
          <tr style = "line-height:30px;border-bottom: 1px solid #ff8800">
          <td style = "font-weight:bold;font-size:24px;text-align:center;">
          </td><td>「批准人」<?php echo $reply['checker']?></td></tr>
    </table>
    <?php }?>
    <?php endforeach;?>
  …
```

代码中直接使用 Wm_Model_Reply 模型方法 getReply 获取事务的批复详情，这也不符合 MVC 的程序设计理念，在后面的小节中将用视图助手完成这项功能。

10.4 事务信息的添加

事务信息的添加与前面章节中所介绍的用户信息、文件信息等的添加是基本相同的，只是对应的数据表不同而已。在前面的各种信息添加过程中，我们使用的是 Zend_Form 表单对象，对它的使用相信大家都已经非常熟悉了，下面使用 Zend Framework 的 Zend_View 组件提供的一种新方法（即“视图助手”）来实现事务信息的添加。

10.4.1 事务信息添加方法的创建

启动 Zend Studio 集成开发环境，选择项目工作区中的 wmProject 项目，通过菜单打开 ZF Tool 工具窗口，输入命令：

```
zf create action add Affair
```

在事务信息管理控制器 Affair 中创建 add 方法，并在方法中添加如下代码：

```
public function addAction()
{
    if($this->getRequest()->isPost()){
        $data = $this->getRequest()->getPost();
        $affair = array(
                'title' => $data['title'],
                'content' => $data['body'],
                'typeid' => $data['typeid'],
                'uid' => $this->_user->getIdentity()->id,
                'checkid' => $data['typeid'],
                'cdat' => time(),
                'state' => 1
        );
        $this->_affair->addAffair($affair);
        $this->redirect('/affair/list/state/1');
    }
```

```
        $modeUser = new Wm_Model_User();
        $users = $modeUser->getUsers("role != 'user'");
        foreach ($users as $checker){
            $arrayName[] = $checker['username'];
            $arrayId[] = $checker['id'];
        }
        $checkers = array_combine($arrayId, $arrayName);
        $this->view->checkers = $checkers;
    }
```

当用户单击事务管理页面中的“新增”按钮后，页面跳转至 Affair 控制器的 add 方法，并渲染 add.phtml 视图，页面效果如图 10.5 所示。

从图中可以看出，表单中的事务“类型”、“审批人”是通过选择确定的，页面显示时就应该对选项进行初始化。为了简单，这里的“类型”直接在视图中赋予固定值；“审批人”从数据库中读入。上述代码的倒数第 2 行至倒数第 8 行完成“审批人”的查询与数组准备工作。事务审批人从用户表中查询得到，他们是拥有审批权限的一批人。由于数据表 tb_affair 中的 checkid 字段存储的是 tb_user 表中的 id，而页面中显示的是审批人的姓名，所以在程序中重新构造了一个名为 checkers 的数组，它的“键”为审批人的 id，它的“值”为审批人的姓名，用这个数组对页面中的“审批人”表单元素进行初始化。

如果用户填写完毕各项表单内容，单击“提交”按钮，数据也被提交到 add 方法中。这时，程序首先接收表单数据，然后依据 tb_affair 数据表的字段补齐表单中缺少的各项字段值，最后调用 Wm_Model_Affair 中的 addAffair 方法将事务信息写入数据表中。

10.4.2 事务信息添加视图的设计

Zend Framework 的 Zend_View 组件是用来在 MVC 模式中处理 View(视图)部分的一个类，是用来使 View、Model 及 Controller 部分的代码分离的。它提供了 helper、output filter、variable escaping 等几个功能组件，其中的 helper 就是“视图助手”(View Helper)，下面用它来编写添加事务信息的表单视图。

打开视图文件 application\views\scripts\affair\add.phtml，并编写代码：

```
…
<div id="msmain">
    <form action="/affair/add" method="post"
          OnSubmit="return check(this)">
      <table class="sheet">
      <tr><th>标题</th><td>
          <?php echo $this->formText('title',null,array(
                                  'size'=>50,'style'=>'padding:5px'))?></tr>
      <tr><th>类型</th><td>
          <?php echo $this->formSelect('typeid','请示',array(
                'style'=>'width:80px;height:25px;margin:5px;padding:3px'),
                array('3'=>'请示','4'=>'报告','5'=>'意见'))?></td></tr>
      <tr><th>内容</th><td>
          <?php echo $this->formTextarea('body',null,array(
                'rows'=>8,'cols'=>65,'style'=>'margin:2px'))?></td></tr>
      <tr><th>审批人</th><td>
```

```
            <?php echo $this->formSelect('checkerid',null,
                  array('style'=>'width:120px;height:25px;
                  margin:5px;padding:3px'),$this->checkers)?></td></tr>
       <tr><th> </th><td>
            <?php echo $this->formSubmit(null,'提交',array(
                  'style'=>'width:50px;height:25px;;margin:5px'))?></td></tr>
          </table>
     </form>
…
<script type="text/javascript">
<!--
function check(theform)
{
    if(theform.title.value == "")
    {
        alert("请输入标题!");
         theform.title.focus();
         return false ;
    }
      if(theform.body.value == "")
    {
        alert("请输入内容!");
         theform.body.focus();
         return false ;
    }
    return true ;
  }
//-->
</script>
```

这里为了简单,没有添加表单数据验证代码,只是简单地用了一个 JavaScript 函数 check 来提示用户填写“标题”及“内容”。

注意:代码中表单元素的创建方法,它既不是 Zend_Form 表单对象,也不是传统的 HTML 表单元素,它们是 Zend_View 视图对象 $this 调用基本视图助手方法来完成的。代码中的 formText、formSelect、formSubmit 为 Zend_View 自带的 helper 类的方法,它们的功能及使用将在下一节介绍。

10.5 Zend_View 视图助手

在 Zend Framework 项目的开发过程中,经常需要在视图中执行某些复杂功能,例如格式化日期,生成表单对象,或者显示 action 的链接等。这些工作都可以使用视图助手类完成,就像 10.4.2 节中所做的那样。

10.5.1 基本视图助手类

所谓“助手”,就是 Zend Framework 定义或用户自定义的一些简单的类,与一般的类不同的是,助手类的名称及方法的定义具有某种特定的要求。

打开 Zend Framework 库文件的 View 目录，可以看到里面有一个名为 Helper 的文件夹，这里存放的就是一些基本的视图助手类，如图 10.8 所示。

Navigation	Partial	Placeholder	Abstract.php
Action.php	BaseUrl.php	Currency.php	Cycle.php
DeclareVars.php	Doctype.php	Fieldset.php	Form.php
FormButton.php	FormCheckbox.php	FormElement.php	FormErrors.php
FormFile.php	FormHidden.php	FormImage.php	FormLabel.php
FormMultiCheckbox.php	FormNote.php	FormPassword.php	FormRadio.php
FormReset.php	FormSelect.php	FormSubmit.php	FormText.php
FormTextarea.php	Gravatar.php	HeadLink.php	HeadMeta.php
HeadScript.php	HeadStyle.php	HeadTitle.php	HtmlElement.php
HtmlFlash.php	HtmlList.php	HtmlObject.php	HtmlPage.php
HtmlQuicktime.php	InlineScript.php	Interface.php	Json.php
Layout.php	Navigation.php	PaginationControl.php	Partial.php
PartialLoop.php	Placeholder.php	RenderToPlaceholder.php	ServerUrl.php
Translate.php	Url.php	UserAgent.php	

图 10.8　基本视图助手的文件目录

打开图 10.8 所示目录中的 FormText.php 文件，内容如下：

```
…
require_once 'Zend/View/Helper/FormElement.php';
…
class Zend_View_Helper_FormText extends Zend_View_Helper_FormElement
{
    …
    public function formText($name, $value = null, $attribs = null)
    {
        $info = $this->_getInfo($name, $value, $attribs);
        extract($info); // name,value,attribs,options,listsep,disable
        // build the element
        $disabled = '';
        if ($disable) {
            // disabled
            $disabled = ' disabled="disabled"';
        }
        $xhtml = '<input type="text"'
                . ' name="' . $this->view->escape($name) . '"'
                . ' id="' . $this->view->escape($id) . '"'
                . ' value="' . $this->view->escape($value) . '"'
                . $disabled
                . $this->_htmlAttribs($attribs)
                . $this->getClosingBracket();

        return $xhtml;
    }
}
```

这是一个名为 Zend_View_Helper_FormText 的类的定义，它拥有一个名为 formText 的公有成员函数。调用该成员函数，将会返回一段 XHTML 代码，很明显这就是我们非常熟悉的“文本输入框”表单元素代码。这个文件实际上就是 formText 视图助手的定义。

从上述代码可以看出，Zend Framework 中助手的定义规则如下：

(1) 类的前缀采用 Zend_View_Helper_(默认助手路径)形式，类名的最后一部分是助手的名称。

(2) 类应当且至少有一个以助手名命名的方法，方法名采用驼峰格式，即首字母小写，之后的每个单词首字母大写，方法的访问权限为 public。

Zend_View 的基本视图助手大部分都是用来生成组件和自动转义变量的，也有些助手用来创建基于路由的 URL 和 HTML 列表以及声明变量，常用的助手名称及功能见表 10.3。

表 10.3　基本视图助手

视图助手	说　明
declareVars()	声明 View 对象中的变量
fieldset($ name, $ content, $ attribs)	生成一个 XHTML fieldset
form($ name, $ attribs, $ content)	生成一个 XHTML 表单
formButton($ name, $ value, $ attribs)	生成<button />元素
formCheckbox($ name, $ value, $ attribs, $ options)	生成<input type="checkbox" />元素
formErrors($ errors, $ options)	生成一个无顺序的 XHTML 列表来显示错误
formFile($ name, $ value, $ attribs)	生成<input type="file" />元素
formHidden($ name, $ value, $ attribs)	生成<input type="hidden" />元素
formMultiCheckbox($ name, $ value, $ attribs, $ options, $ listsep)	生成一个 checkboxes 列表
formPassword($ name, $ value, $ attribs)	生成<input type="password" />元素
formRadio($ name, $ value, $ attribs, $ options)	生成一系列<input type="button" />元素
formReset($ name, $ value, $ attribs)	生成<input type="reset" />元素
formSelect($ name, $ value, $ attribs, $ options)	生成<select>... </select>，其中的每个<option>对应一个 $ option 数组元素
formSubmit($ name, $ value, $ attribs)	生成<input type="submit" />元素
formText($ name, $ value, $ attribs)	生成<input type="text" />元素
formTextarea($ name, $ value, $ attribs)	生成<textarea>... </textarea>元素
action($ action, $ controller, $ module = null, array $ params=array())	动作视图助手，执行一个特定的控制器 Action
partial	区域助手，用来在它自己的变量范围内解析特定的模板
partialLoop	区域助手，允许传递可迭代数据并为每个条目解析
placeholder	占位符助手，在视图脚本和视图实例之间持久化内容
Doctype	指定 HTML 或 XHTML 文档类型
HeadLink	生成<link>元素
HeadMeta	生成<meta>元素，提供关于 HTML 文档的 meta 信息
HeadStyle	生成<head>元素中的 HTML <style>元素
HeadTitle	生成<head>元素中的 HTML <title>元素

表中列出的大部分基本视图助手(如 Doctype、HeadTitle、partialLoop 等)在前面的章节中已经使用过，这里不再赘述，请大家结合 Zend Framework 库源文件及手册慢慢体会。

10.5.2 自定义视图助手

Zend_View 组件的基本视图助手数量是有限的，并且功能比较单一，在项目的开发过程中还需要创建一些适合自己应用需求的视图助手类。

有了上面对 Zend_View 中基本视图助手的分析，了解了它们的基本结构后，编写自定义的助手类就非常容易了，只要遵循以下几个原则即可：

(1) 助手类的类名的最后部分必须是助手的名称，并使用驼峰格式。

例如要定义一个名为 affairState 的助手类，类名的最后部分应为 AffairState，前缀为 Zend_View_Helper_，所以，类的全名应为 Zend_View_Helper_ AffairState。

(2) 类中必须有一个 public 方法，该方法名与助手类名相同。

该方法将在调用 $ this-> affairState()时执行。在(1)中的例子中，相应的方法声明应该是 public function affairState()。注意方法名的第 1 个字母为小写。

(3) 一般来说，助手类不应该有 echo、print 或其他形式的输出，只需要返回值就可以了，并且返回的数据应当被转义。

(4) 类文件的命名应该是助手类的名称，例如在(1)中的例子中文件要存为 AffairState. php。

(5) 将助手类的文件放在 application\views\helpers 目录下。这样，Zend_View 将会自动加载、实例化、持久化、并执行视图助手。

10.5.3 事务信息视图的优化

1. 定义视图助手类

启动 Zend Studio 集成开发环境，选择项目工作区中的 wmProject 项目，在 application\views\helpers 目录上右击鼠标，打开新建 PHP 文件对话框，创建名为 AffairState. php 的文件，并添加代码：

```
<?php
class Zend_View_Helper_AffairState
{
  public function affairState( $ state)
  {
      switch ( $ state) {
          case 1: $ str = '[审核中...]';break;
          case 2: $ str = '[等待办理...]';break;
          case 3: $ str = '[正在办理...]';break;
          case 4: $ str = '[已经完成]';break;
          default: $ str = '';break;
      }
      return htmlspecialchars( $ str);
  }
}
```

2. 修改视图文件代码

```
<?php foreach ( $ this -> affairs as $ affair) :?>
```

```
<table style="background:url(/images/date-bg2.gif)
                                no-repeat left top;margin-top:5px">
    <tr><td width="60px" align="center">
            <?php echo date('m', $affair['cdat'])?>月</td>
    <td style="line-height:30px;border-bottom: 1px dashed #ff8800">
    "标题"<?php echo $affair['title']?>  <font color='red'>
    <?php echo $this->AffairState($affair['state']) ?>
    </font></td></tr>
     …
```

在视图代码中,可以调用 AffairState 助手任意次。它只被实例化一次,并且会在 Zend_View 实例的生命周期内持久存在。

10.6 本章小结

本章介绍系统事务信息管理功能的实现,包括数据库的设计、事务的添加、事务的分类显示、事务提醒、批示回复以及事务处理状态跟踪等。事务的添加采用 Zend Framework 的视图助手来完成,这是一种 Web 表单处理的新的解决方案。

本章重点掌握办公自动化管理系统中的事务信息管理的业务逻辑,以及 Zend Framework 框架的视图助手的概念、常用助手的使用、自定义助手的创建规则。

第 11 章 日常办公常用功能模块

办公自动化管理系统在企业信息管理中的应用不仅规范了企业的行政程序，而且大大提高了办公效率，使企业员工的工作责任感、紧迫感得到了进一步增强。效率的提高意味着员工劳动强度的增大、工作压力的增加。为了营造一个快捷、高效、轻松的在线办公环境，本系统设置了一些常用的辅助办公功能。例如，为员工设置个人网络存储空间、允许用户制订个人工作日程与撰写个人工作日志、提供万年历、提供通讯录等。

本章实现办公自动化管理系统的办公常用功能，包括网络 U 盘、日程信息管理、日历查询等，由此介绍 Zend Framework 的 Zend_Date 组件，并进一步熟悉 Zend_View 视图助手、文件上传及其他组件的使用。

11.1 日常办公常用功能效果预览

本系统实现了 3 种常用的在线办公功能，即用户网络空间、用户日程信息管理及日历查询。“网络空间”模块方便用户资料的分类管理与传递，它是资源共享的一个重要方面；“日程信息管理”模块允许用户制订个人工作计划、撰写个人工作日志，以增强用户工作中的条理性与归纳性，从而提高工作质量；“日历查询”模块既属于通用查询功能，同时也为其他功能模块提供服务。

11.1.1 用户网络空间页面效果

1. 网络空间根目录页面

图 11.1 所示的是系统为用户提供的网络空间主页面效果。在线办公系统需要给每一位登录用户提供一个网络空间，空间的大小由系统管理员决定。它可以由系统管理员统一设置，也可以由用户自己创建，本系统采用的是用户自定义创建的方式。

用户登录后，选择主菜单中的“个人办公”|“网络存储”菜单项或快捷菜单中的“我的办公桌”|“网络 U 盘”菜单项均可进入用户的个人网络空间。如果是第 1 次进入该页面，系统会提示用户创建自己的网络空间根目录，这里根目录的名字固定为用户姓名。

从图 11.1 所示的页面效果可以看出，用户网络空间界面分为上、下两个区域，上部区域为操作功能区，包括“向上一级”、“根目录”、“新文件夹”、“上传文件”、“剪切”、“粘贴”和“打包下载”7 种操作；下部区域为目录内容显示区，以列表的形式输出当前目录下的子文件夹及文件。另外，页面的顶部还设有网络空间大小及使用容量的说明。

2. 网络空间子目录页面

图 11.2 所示为用户单击子文件夹后的页面效果。在用户网络空间的显示区域中，子目



图 11.1　用户网络空间根目录页面

录名及文件名均为超级链接。用户如果单击子目录名，则进入该子目录；如果单击文件名，则会打开该文件或进入文件的下载页面。内容显示区的表头是用户网络空间的目录层次结构。



图 11.2　用户网络空间子目录页面

3. 新建文件夹页面

图 11.3 所示为用户单击操作区中的"新建文件夹"按钮后的页面效果。用户单击"新建文件夹"按钮后，在页面的操作区与显示区之间出现创建文件夹表单，用来接收用户输入的新文件夹的名字。

4. 上传文件页面

图 11.4 所示为用户单击操作区中的"上传文件"按钮后的页面效果。用户单击"上传文件"按钮后，在页面的操作区与显示区之间出现文件上传表单，方便用户选择需要上传的文件。



图 11.3　新建文件夹页面



图 11.4　上传文件页面

11.1.2　用户日程信息管理页面效果

1. 日程信息管理主页面

图 11.5 所示为用户日程信息管理主页面效果。用户登录后，选择主菜单中的“个人办公”|“个人日程”菜单项或快捷菜单中的“我的办公桌”|“日程安排”菜单项即可进入用户的日程信息管理主页面。

用户日程信息管理模块借用了第 10 章中的事务信息管理视图结构，它的导航菜单包含“今日工作”、“本周安排”、“日程查询”与“待办提醒”4 个子项。这里的主页面也是“今日工



图 11.5　用户日程信息管理页面

作”的显示页面。

2. 本周安排页面

图 11.6 所示为用户本周工作安排页面效果，在主显示区以表格的形式详细列出了本星期的每一个工作日需要完成的工作任务。



图 11.6　用户本周安排页面

3. 日程查询页面

图 11.7 所示为用户日程查询页面效果，这里以电子台历的表现形式完成用户的日程查询功能，符合用户的工作与生活习惯。用户单击台历中的某一天，即可了解该日的工作安排详情。

4. 添加事务页面

图 11.8 所示为用户制订日程时添加事务的页面效果，表单中设有“事务类型”、“标题”、“内容”、“开始日期”和“提醒日期”5 种表单元素，用于接受用户的输入。事务类型分为“个人事务”、“工作事务”；事务开始日期为某项安排开始实施的时间，由用户单击日历图标进



图 11.7　用户日程查询页面

行选择，如图 11.9 所示；事务提醒日期为该项工作安排的系统提示开始时间，为了简单，这里固定为事务开始时间的前 3 天。



图 11.8　添加事务页面



图 11.9　设置事务开始时间页面

11.2 数据库设计

本章主要实现用户网络空间与日程信息管理的功能，根据第 3 章的总体设计与功能模块的业务逻辑创建"用户文件夹"、"用户文件"和"用户日程"3 张数据表存储功能实现中的数据信息。

11.2.1 用户网络空间模块数据库

创建用户网络空间功能模块数据库，首先必须设计实现该功能的业务流程，也就是业务逻辑。在图 11.1 所示的页面中，大家看到了一个类似"资源管理器"的文件系统界面，它具有清晰的目录层次结构。然而，这里呈现的目录层次结构实际上是虚拟的，在服务器上并不存在这么多的子文件夹，所有用户上传文件都存放在一个以用户名命名的文件夹中，如图 11.10 所示。该图中的"微梦"与"微梦网"文件夹即为登录用户网络空间的根目录。子文件夹的层次结构以及"剪切"、"粘贴"等操作其实是通过数据库表实现的。当然，大家也可以在服务器上创建真实的子目录层次结构。

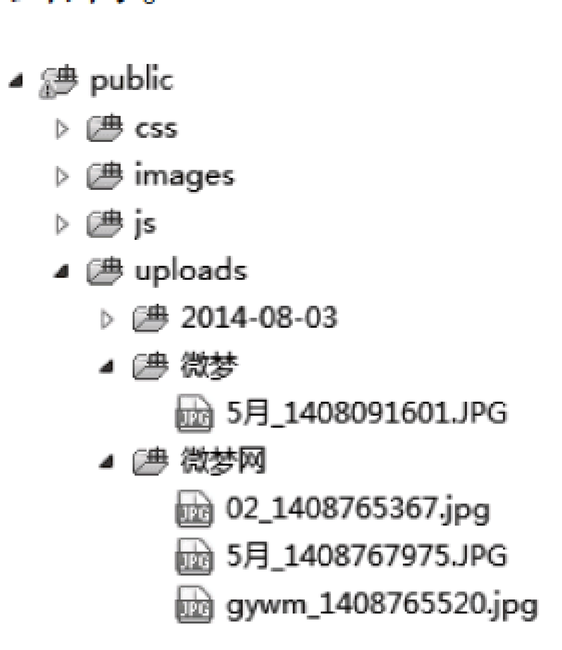


图 11.10 用户网络空间目录结构

根据本系统用户网络空间的创建与操作业务逻辑，在数据库中创建"用户文件夹"tb_userfolder 以及"用户文件"tb_userfile 两张数据表。

1. tb_userfolder 数据表

打开 phpMyAdmin 数据库管理工具，在 db_wmoams 数据库中新建名为 tb_userfolder 的数据表，其字段含义见表 11.1。字段属性如图 11.11 所示。

表 11.1 tb_userfolder 数据表的字段说明

字段名	说 明
id	自增变量，主键，目录的唯一标识
foldername	文件夹名称
fid	文件夹父目录 ID，根目录为 0
layerid	目录层次，根目录为 1
uid	目录创建人 ID，对应 tb_user 表的 id 字段
cdat	目录创建时间
share	是否共享，0 为非共享、1 为共享

名字	类型	排序规则	属性	空	默认	额外
id	bigint(20)			否	无	AUTO_INCREMENT
foldername	varchar(20)	gbk_chinese_ci		否		
fid	bigint(20)			否	0	
layerid	tinyint(6)			否	0	
uid	int(11)			否	0	
cdat	int(11)			否	0	
share	tinyint(1)			否	0	

图 11.11　tb_userfolder 数据表的字段属性

文件夹数据表 tb_userfolder 设有 7 个字段，其中 fid 表示当前目录的上一级父目录 ID，它的值应为本数据表中字段 id 存储的数据，用户网络空间根目录的 fid 为 0；字段 layerid 表示当前目录的层次，用于“向上一级”操作功能的实现。用户每单击一次“向上一级”操作按钮，layerid 在当前值基础上减 1，直到它的值变为 1，即达到用户网络空间根目录为止。表中的 share 字段用来设置目录的共享属性。

2. tb_userfile 数据表

打开 phpMyAdmin 数据库管理工具，在 db_wmoams 数据库中新建名为 tb_userfile 的数据表，其字段含义见表 11.2。字段属性如图 11.12 所示。

表 11.2　tb_userfile 数据表的字段说明

字　段　名	说　　明
id	自增变量，主键，文件的唯一标识
filename	文件名称
folderid	文件所属文件夹 ID，对应 tb_userfolder 表的 id 字段
userid	文件创建人 ID，对应 tb_user 表的 id 字段
size	文件大小
extend	文件扩展名
cdat	文件创建时间

名字	类型	排序规则	属性	空	默认	额外
id	bigint(20)			否	无	AUTO_INCREMENT
filename	varchar(100)	gbk_chinese_ci		否		
folderid	int(11)			否	0	
userid	int(11)			否	0	
size	int(11)			否	无	
extend	varchar(6)	utf8_general_ci		否		
cdat	int(11)			否	0	

图 11.12　tb_userfile 数据表的字段属性

文件数据表 tb_userfile 也设有 7 个字段，其中 folderid 表示文件所属目录 ID，它对应 tb_userfolder 数据表中的 id 字段；size 表示文件大小，单位为 bit，用于计算用户网络空间的使用情况；extend 表示文件扩展名，用于文件显示时图标类型的确定。在如图 11.2 所示的页面中，文件名称前面都带有一个表示文件类型的图标，这些图片的文件名是用相应的文件扩展名来命名的。例如，Microsoft Word 文件的图标名称为 doc.gif。

11.2.2 用户日程信息管理模块数据库

用户日程信息管理功能的实现过程与第 10 章介绍的事务信息管理模块基本相同，主要是日程的添加与事务的分类显示。不同的是，用户日程信息管理模块中设有“待办提醒”功能，所以在数据库中必须为每项安排设置系统提醒的开始时间。

打开 phpMyAdmin 数据库管理工具，在 db_wmoams 数据库中新建名为 tb_schedule 的数据表，其字段含义见表 11.3。字段属性如图 11.13 所示。

表 11.3 tb_schedule 数据表的字段说明

字 段 名	说 明
id	自增变量，主键，日程事务的唯一标识
uid	日程创建人 ID，对应 tb_user 表的 id 字段
title	日程事务标题
content	日程事务内容
cdat	日程事务创建时间
bdat	日程事务实施时间
rdat	日程事务系统提醒开始时间
typeid	日程事务类型
state	日程事务处理状态

名字	类型	排序规则	属性	空	默认	额外
id	int(11)			否	无	AUTO_INCREMENT
uid	tinyint(4)			否	0	
title	varchar(80)	gbk_chinese_ci		否		
content	varchar(250)	gbk_chinese_ci		否		
cdat	int(11)		UNSIGNED	否	无	
bdat	int(11)			否	无	
rdat	int(11)			否	无	
typeid	tinyint(4)			否	0	
state	tinyint(4)			否	0	

图 11.13 tb_schedule 数据表的字段属性

日程信息数据表 tb_schedule 设有 9 个字段，与事务数据表 tb_affair 基本相似。其中的 typeid 表示日程事务的类型，这里仅设“工作事务”与“个人事务”两种类型；state 表示日程事务的处理进度。

11.3 用户网络空间功能模块

通过上面两节的学习，我们熟悉了用户网络空间功能模块的框架结构、业务处理逻辑，并且完成了数据库的设计，下面编码实现模块功能。

11.3.1 控制器及方法的创建

为了让用户拥有自己的网络空间，并能在空间中进行文件夹及文件的相关操作，需要设置相应的控制器及方法，控制器及方法的创建还是采用 Zend Studio 集成开发环境提供的

Zend Framework 工具来完成。

1. 创建控制器及方法

启动 Zend Studio 集成开发环境，选择项目工作区中的 wmProject 项目，然后选择 Project 主菜单中的 Zend Tool 菜单项，打开 ZF Tool 工具窗口，输入命令：

```
zf create controller Netdisk
```

在默认模块 default 中创建网络空间信息管理控制器 Netdisk。然后用同样的方法打开 ZF Tool 工具窗口，分别输入命令：

```
zf create action create Netdisk
zf create action addfolder Netdisk
zf create action list Netdisk
```

在控制器 Netdisk 中创建 create、addfolder 和 list 3 个方法。create 方法用来创建用户网络空间；addfolder 方法用来创建新的文件夹；list 方法负责数据的显示。

2. 初始化控制器

利用 Zend Framework 的命令工具生成控制器及方法的框架代码后，需要在框架文件中编写自己的业务逻辑代码。为了编程方便，在 Netdisk 控制器中定义 user、ufolder、ufile 3 个对象，分别表示登录用户、目录模型、文件模型，并在 init 方法中对其初始化。代码如下：

```
class NetdiskController extends Zend_Controller_Action
{
    protected $_ufolder = null;
    protected $_ufile = null;
    protected $_user = null;
    public function init()
    {
        $auth = Zend_Registry::get('auth');
        if($auth->hasIdentity()){
            $this->_user = $auth;
        }
        $this->_ufolder = new Wm_Model_Ufolder();
        $this->_ufile = new Wm_Model_Ufile();
    }
…
}
```

对于对象的访问权限，这里使用的是 pretected，也可以使用 private 的私有访问方式。代码中的 Wm_Model_Ufolder 与 Wm_Model_Ufile 为 11.2 节中创建的 tb_userfolder、tb_userfile 数据表的表模型，我们将在接下来的小节中创建它们。用户信息从用户认证信息对象 auth 中提取得到。

11.3.2 数据表模型及方法的设计

在上面的控制器类文件中创建了“文件夹表”模型对象_ufolder 和“文件表”模型对象_ufile，并在初始化方法中对其进行了初始化。通过这两个对象可以对“文件夹表”和“文件

表”实施增、删、改、查等各种操作。

1. 创建模型

打开 Zend Studio 集成开发环境中的 ZF Tool 工具窗口，分别输入命令：

```
zf create model Ufolder
```

和

```
zf create model Ufile
```

创建 Wm_Model_Ufolder 与 Wm_Model_Ufile 模型类。为类添加基类及对应的数据表名，代码如下：

```
class Wm_Model_Ufolder extends Zend_Db_Table
{
    protected $_name = 'tb_userfolder';
   …
}

class Wm_Model_Ufile extends Zend_Db_Table
{
  protected $_name = 'tb_userfile';
    …
}
```

2. 添加方法

打开模型文件 application\models\Ufolder.php，为表模型类 Wm_Model_Ufolder 添加 createDisk 方法、getUfolder 方法及 getUfolders 方法，代码如下：

```
public function createDisk($data = array())
{
     $row = $this->createRow();
     if (count($data) > 0){
         foreach ($data as $key => $value){
              $row->$key = $value;
         }
         $row->save();
         return $row->id;
     }
     else{
         throw new Zend_Exception('申请网络 U 盘失败!');
     }
}

public function getUfolder($where)
{
     if (is_numeric($where)){
          $row = $this->find($where)->current();
     }
     if (is_array($where)){
          $select = $this->select();
```

```
            if (count( $ where) > 0){
                foreach ( $ where as  $ key => $ value){
                    $ select -> where( $ key. " = ?", $ value);
                }
            }
            $ row = $ this -> fetchRow( $ select);
        }
        if ( $ row){
            return  $ row;
        }
        else{
            return null;
        }
    }

    public function getUfolders( $ where = array(), $ order = null)
    {
        $ select = $ this -> select();
        if (count( $ where) > 0){
            foreach ( $ where as  $ key => $ value){
                $ select -> where( $ key. ' = ?', $ value);
            }
        }
        if ( $ order){
            $ select -> order( $ order);
        }
        $ result = $ this -> fetchAll( $ select);
        if ( $ result){
            return  $ result;
        }
        else{
            return null;
        }
    }
```

上述代码与前面几章中模型方法的代码相似，非常容易理解。createDisk 方法用来创建用户的网络空间，并返回新空间根目录的 ID；getUfolder 方法调用 fetchRow 函数获取单个文件夹信息，返回结果为 Zend_Db_Table_Row_Abstract 对象；getUfolders 方法调用 fetchAll 函数获取某一类文件夹信息，返回结果为 Zend_Db_Table_Rowset_Abstract 对象。模型方法调用后的返回值类型关系到数据在视图中的显示方法。

完成 Wm_Model_Ufolder 表模型的设计后，接下来为表模型 Wm_Model_Ufile 添加 addFile 和 getUfiles 方法，代码如下：

```
    public function addFile( $ data = array())
    {
        $ row = $ this -> createRow();
        if (count( $ data) > 0){
            foreach ( $ data as  $ key => $ value){
                $ row -> $ key = $ value;
```

```
            }
            $ row -> save();
            return $ row -> id;
        }
        else{
            throw new Zend_Exception('添加文件失败!');
        }
    }
    public function getUfiles( $ where = array(), $ order = null)
    {
         $ select = $ this -> select();
        if (count( $ where) > 0){
            foreach ( $ where as $ key => $ value){
                $ select -> where( $ key. ' = ?', $ value);
            }
        }
        if ( $ order){
            $ select -> order( $ order);
        }
         $ result = $ this -> fetchAll( $ select);
        if ( $ result){
            return $ result;
        }
        else{
            return null;
        }
    }
```

11.3.3 自定义视图助手

在如图 11.1 所示的用户网络空间页面中需要输出空间的使用情况以及当前目录的层次结构，我们用自定义的视图助手来完成这两项功能。

1. NetDiskSize 视图助手

启动 Zend Studio 集成开发环境，选择 wmPoject 目录下的 application\views\helpers 文件夹，然后右击该文件夹，通过快捷菜单打开新建 PHP 文件对话框，创建名为 NetDiskSize.php 的文件，并添加代码：

```
class Zend_View_Helper_NetDiskSize
{
  public function netDiskSize()
  {
        $ auth = Zend_Registry::get('auth');
        $ userId = $ auth -> getIdentity() -> id;
        $ modelFile = new Wm_Model_Ufile();
        $ files = $ modelFile -> getUfiles(array('userid' => $ userId));
        $ sizes = 0;
       foreach ( $ files as $ file){
            $ sizes += $ file['size'];
       }
```

```
        return $sizes;
    }
}
```


根据第 10 章中介绍的视图助手类的定义规则，该视图助手类的类名为 NetDiskSize，与文件名保持一致；其前缀采用默认形式 Zend_View_Helper_，类中有一个与类名相同的 public 方法。

该视图助手用来返回用户网络空间的容量使用情况。程序首先获取注册表中的用户认证对象，提取当前用户的 ID，然后调用数据表 tb_userfile 的模型方法 getUfiles，得到属于当前用户的全部文件，最后遍历所有文件对象，将文件大小相加，即可得到当前用户的网络空间已使用的容量值。

2. FolderPosition 视图助手

使用上面的方法在 helpers 文件夹中添加新文件 FolderPosition.php，并在文件中添加如下代码：

```
class Zend_View_Helper_FolderPosition
{
  public function folderPosition($foldeId)
  {
        $modelFolder = new Wm_Model_Ufolder();
        $fatherFolder = $modelFolder->getUfolder($foldeId);
        $fid = $fatherFolder['fid'];
        $str = $fatherFolder['foldername'];
        if ($fid){
            while ($fid){
                $fatherFolder = $modelFolder->getUfolder($fid);
                $str = $fatherFolder['foldername'].'&gt;&gt; '.$str;
                $fid = $fatherFolder['fid'];
            }
        }
        return $str;
  }
}
```

该视图助手用来输出用户网络空间当前目录的层次结构字符串，如图 11.3 所示。程序首先通过当前文件夹的 ID 调用 Wm_Model_Ufolder 模型的 getUfolder 方法，得到当前目录名称及其父目录 ID，即 fid；然后递归检测父目录的 fid，直到 fid 等于 0，即用户网络空间的根目录为止。

11.3.4 创建用户网络空间

在前面已经说过，本系统的用户网络空间需要用户申请。因此，用户第 1 次进入网络空间信息管理模块时，出现的页面界面并不是如图 11.1 所示的效果，而是如图 11.14 所示的结构。页面中设置的"单击这里开通网络 U 盘"超级链接，就是用于网络空间创建的。除此之外，页面中还有当前用户可以申请的网络空间大小的说明。

图 11.13 所示的页面为 Netdisk 控制器的 index.phtml 视图效果之一，下面在 index 方



图 11.14 用户开通网络空间页面

法及其视图文件中添加代码。

1. 编写 index 方法代码

打开 Netdisk 控制器文件 application\controllers\NetdiskController.php，在其 index 方法中添加如下代码：

```
public function indexAction()
{
     $userid = $this->_user->getIdentity()->id;
     $where = array('uid' => $userid, 'fid' => 0);
     $isfolder = $this->_ufolder->getUfolder($where);
    if($isfolder == null){
         $this->render();
    }else{
        $rootfolder = $isfolder->toArray();
        $this->redirect('/netdisk/list/folderid/'.$rootfolder['id']);
    }
}
```

程序首先从控制器对象 user 中获取当前用户 ID，然后查找用户网络空间根目录。如果根目录存在，则跳转到 list 方法进入用户网络空间；否则说明用户还未申请网络空间，页面转向图 11.13 所示的界面，提醒用户开通自己的网络空间。

2. 编写 index 方法视图代码

打开 application\views\scripts\netdisk\index.phtml 视图文件，添加如下代码：

```
<?php require_once 'netdiskcss.php';?>
<div id="dtitle" align="center">用户网络 U 盘</div>
<table width="90%" border="0" cellpadding="0" cellspacing="0">
  <tr><td>
```

```
    <table width="100%" border=0 align=center cellPadding=2 cellSpacing=1 class=
tableborder>
        <tbody><tr><td class=tablerow align=left>
        您还没有自己的网络U盘,请单击下面的链接申请!您的网络U盘的最大容量为100MB
        </td></tr></tbody>
    </table>
</td></tr><tr><td>
<table cellPadding="2" cellSpacing=1 class=tableborder>
<tbody id=tbody1><tr><td valign="middle">
<a href="/netdisk/create"><b>单击这里开通网络U盘</b></a>
</td></tr></tbody></table>
</table>
```

为了简单,这里分配给每个用户的网络空间均为100MB,在实际应用中该项指标应根据用户角色或权限指定,并设置相应的开通、锁定及删除等管理功能。代码中的阴影部分为用户申请网络空间超级链接。

3. 编写 create 方法代码

从上面的 index.phtml 视图代码中可以看到,当用户单击“开通网络U盘”超级链接后,程序跳转到 Netdisk 控制器的 create 方法,下面是该方法的代码:

```
public function createAction()
{
        $userid = $this->_user->getIdentity()->id;
        $username = $this->_user->getIdentity()->username;
        $data = array('foldername' => $username.'(U盘根目录)',
                'layerid' => 1,
                'fid' => 0,
                'uid' => $userid,
                'cdat' => time()
        );
        $folder = $this->_ufolder->createDisk($data);
        $this->redirect('/netdisk/list/folderid/'.$folder);
}
```

程序首先获取当前用户的ID及姓名,然后确定根目录的名称、层级、父目录、创建人及创建日期,并把它们赋与data数组,最后调用Wm_Model_Ufolder表模型的createDisk方法将根目录数据写入到tb_userfolder数据表中,完成用户网络空间的创建。

11.3.5 显示用户网络空间

用户网络空间的显示功能由Netdisk控制器的list方法完成。用户网络空间的显示请求可能来自主菜单、快捷菜单、“向上一级”按钮、“根目录”按钮以及子文件夹名称等超级链接,这些请求中所传递的参数往往是不同的,所以在list方法中要考虑各种请求的参数传递形式。

1. 编写 list 方法代码

打开Netdisk控制器文件application\controllers\NetdiskController.php,在方法list中添加如下代码:

```
public function listAction()
{
        //当前文件夹
        $folderid = $this->getRequest()->getParam('folderid');
        if(!isset($folderid)){
           $userid = $this->_user->getIdentity()->id;
           $username = $this->_user->getIdentity()->username;
           $where = array('uid' => $userid, 'fid' => 0);
           $pfolder = $this->_ufolder->getUfolder($where)->toArray();
           $rootfolderid = $pfolder['id'];
           $folderid = $pfolder['id'];
        }else{
           $pfolder = $this->_ufolder->find($folderid)->current();
        }
        //当前文件夹的子文件夹
        $folders = $this->_ufolder->getUfolders(array('fid' => $folderid));
        //当前文件夹中的文件
        $files = $this->_ufile->getUfiles(array('folderid' => $folderid));
        $this->view->folders = $folders;
        $this->view->files = $files;
        $this->view->pfolder = $pfolder;
}
```

用户网络空间的显示一定是针对空间根目录或某个子目录的，所以，list 方法的首要任务就是要获取显示目录的 id。如果请求中没有传递需要显示的文件夹的 id，则默认显示用户空间的根目录，上述代码中的 if…else 结构就是实现这个功能的。得到目录的 id 后，就可以调用模型方法、查询该目录下的子文件夹及文件，最后将查询结果传递到视图中。

2. 编写 list 方法视图代码

打开 application\views\scripts\netdisk\list.phtml 视图文件，添加如下代码：

```
<?php require_once 'netdiskcss.php';?>
<div id="dtitle" align="center">用户网络 U 盘</div>
<table width="100%" border="0" cellpadding="0" cellspacing="0" align=center>
<tr><td  valign="top">
     <table cellPadding=2 cellSpacing=1  width="100%" border="1">
         <tbody><tr><td height=24 valign=middle><b>
                    你的网络 U 盘容量是 100MB,已使用的容量:
<?php echo $this->netDiskSize()/1000.0;?> KB
</b></td></tr></tbody></table></td></tr>
<tr><td>
<script type="text/javascript">
    var aID = 0;
    function ShowTabs1(ID){
        if(ID!= aID){
                    Tabs1[aID].style.display = "none";
                    Tabs1[ID].style.display = "";
                    aID = ID;
                    }
    }
</script>
```

```
<table width="500" height="62" border="0" cellPadding=2
                                          cellSpacing=1 class=tableborder>
         <tr>
             <td width="60" align="center" valign="middle" height="40">
<a href="/netdisk/list<?php echo
 $this->pfolder['fid'] != 0 ? '/folderid/'. $this->pfolder['fid'] : '/'?>">
<img src="/images/up.gif" alt="上一级" width="32" height="32"
                                          align="absmiddle" border=0 /></a></td>
             <td width="60" align="center" valign="middle">
                 <a href="/netdisk/list">
                 <img src="/images/home.gif" alt="根目录" border=0 /></a></td>
             <td width="60" align="center" valign="middle">
                  <a href="#" onclick="ShowTabs1(1)">
                  <img src="/images/new_folder.gif"border=0 /></a></td>
             <td width="60"  align="center" valign="middle">
                  <a href="#" onclick="ShowTabs1(2)">
                  <img src="/images/upload.gif" alt="上传" border=0 /></a></td>
             <td width="60"  align="center" valign="middle"><a href="#" >
                  <img src="/images/bcut.gif" alt="剪切"border=0 /></a></td>
             <td width="60"  align="center" valign="middle"><a href="#" >
                  <img src="/images/paste.gif" alt="粘贴" border=0 /></a></td>
             <td width="60"  align="center" valign="middle"><a href="#" >
                  <img src="/images/bzip.jpg" alt="打包下载" /></a></td>
             <td width="150" rowspan="2"></td>
         </tr>
         <tr>
            <td height="22" align="center" width="60">向上一级</td>
            <td align="center" valign="middle" width="60">根目录</td>
            <td align="center" valign="middle" width="60">新文件夹</td>
            <td align="center" valign="middle" width="60">上传文件</td>
            <td align="center" valign="middle" width="60">剪切</td>
            <td align="center" valign="middle" width="60">粘贴</td>
            <td align="center" valign="middle" width="60">打包下载</td>
         </tr>
       </table>
</td></tr>
<tr><td>
<table width="100%" border="0" cellpadding="0" cellspacing="0">
    <tbody style="display:block" id="Tabs1">
         <tr><td height="20"><div id="msg"></div> </td></tr>
    </tbody>
    <tbody style="display:none" id="Tabs1">
           <tr>
                  <td height="20" >文件夹名称:
                  <form method="post" action="/netdisk/addfolder" >
                  <input type="text" name="foldername" size=20 />
                  <input type="hidden" name="folderid"
                                value="<?php echo $this->pfolder['id'] ?>" />
                  <input type="hidden" name="layerid"
                                value="<?php echo $this->pfolder['layerid'] ?>" />
                  <input type="submit" value="创建文件夹"  />
                  </form>
```

```
                </td>
            </tr>
        </tbody>
    <tbody style="display:none" id="Tabs1">
        <tr>
                <td height="20" >文件上传:</td></tr>
                <?php echo $this->action('upload','common',null,
                                array('folderId'=>$this->pfolder['id']))?>
        </tbody>
</table></td></tr>
<tr><td>当前目录:
<?php echo $this->folderPosition($this->pfolder['id']) ?>
</td></tr>
<tr><td>
<table cellPadding=2 cellSpacing=1 width="500" class=tableborder>
<tr>
  <td width="5%">选中</td>
  <td width="35%">名称</td>
   <td width="10%">大小</td>
   <td width="30%">更新时间</td>
  <td width="20%">操作</td>
</tr>
<tbody id=tbody1>
<?php foreach ($this->folders as $folder):?>
<tr bgcolor="#F8F8F8" ><td>
    <input type=checkbox name=checkfolder id=checkfolder value="" >
</td><td>
    <img src="/images/folder.gif" border=0 alt="文件夹" >
    <a href="/netdisk/list/folderid/<?php echo $folder['id']?>">
        <?php echo $folder['foldername']?>
    </a></td>
  <td>-</td>
  <td><?php echo date('Y-m-d H:i:s', $folder['cdat'])?></td>
  <td><a href="#" ><img src="/images/del.gif" border=0 alt="删除" ></a><a href=
"#" ><img src="/images/cut.gif" border=0 alt="剪切" ></a></td>
</tr>
<?php endforeach;?>
</tbody>
<form name="form3" method="post"  action="/netdisk/list" >
      <input type=hidden name=checkLines>
<tbody id=tbody2>
<?php if(count($this->files)):?>
<?php foreach ($this->files as $file):?>
<tr bgcolor="#F8F8F8" id=file>
      <td><input type=checkbox name=checkLine id=checkLine></td>
      <td><img src="/images/img_file/
                <?php echo $file['extend']?>.gif" border=0  alt='文件'>
                <a href="#"><?php echo $file['filename']?></a></td>
      <td><?php echo $file['size'];?></td>
      <td><?php echo date('Y-m-d H:i:s', $file['cdat']);?></td>
      <td><a href="#" ><img src="/images/del.gif" border=0></a>
```

```
            <a href = # >< img src = /images/cut.gif border = 0 >
          </a> </td>
    </tr>
    <?php endforeach;?>
    <?php endif;?>
    </tbody></form></table> </td></tr></table>
```

上述视图代码完成如图 11.1～图 11.4 所示的页面效果。虽然代码比较长，但大部分都是HTML 的页面布局元素，在学习过程中请大家特别注意阴影部分代码的作用。说明如下：

1）用户网络空间使用大小

代码如下：

```
<?php echo $this->netDiskSize()/1000.0;?> KB
```

使用自定义的 NetDiskSize 视图助手获得用户网络空间的使用情况，并将空间大小单位转化成 KB。

2）用户网络空间目录层次结构

代码如下：

```
<?php echo $this->folderPosition($this->pfolder['id']) ?>
```

使用自定义的 FolderPosition 视图助手输出当前目录的层次结构。

3）文件上传

代码如下：

```
<?php echo $this->action('upload','common',null,array('folderId'=>$this->pfolder['id']))?>
```

使用动作助手 action 完成文件上传。这里调用了 Common 控制器的 upload 方法，这个控制器是我们自定义的，将在后面进行介绍。action 动作助手的 4 个参数分别表示控制器方法、控制器、模块与参数，上面的代码相当于/common/upload/folderId/$this—>pfolder['id']请求。

4）返回上一级目录

代码如下：

```
<a href="/netdisk/list<?php echo
 $this->pfolder['fid'] != 0 ? '/folderid/'.$this->pfolder['fid'] : '/'?>">
```

使用条件语句通过当前目录的 fid 值确定转向的页面，根目录的 fid 为 0。

5）子文件夹的显示

代码如下：

```
<?php foreach ($this->folders as $folder):?>
…
<?php endforeach;?>
```

使用循环完成子目录的显示，注意遍历过程中文件夹名称的获取与链接的设置。

6）文件的显示

代码如下：

```
<?php if(count($this->files)):?>
```

```
<?php foreach ( $ this -> files as $ file):?>
< img src = "/images/img_file/<?php echo $ file[ 'extend']?>.gif" />
…
<?php endforeach;?>
<?php endif;?>
```

使用循环完成目录中文件的显示，注意遍历过程中文件名称的获取与文件图标的 src 属性设置。

11.3.6 新建文件夹与上传文件

图 11.1 所示的页面操作区域中显示了 7 种功能按钮，在 list.pthml 视图中已经实现了“上一级目录”与“根目录”功能。由于篇幅的关系，在剩下的 5 种操作中这里只介绍“新建文件夹”与“文件上传”两种功能的实现。

1. 新建文件夹

当用户单击视图页面中的“新文件夹”按钮后，页面中出现新建文件夹表单，该表单提交至 Netdisk 控制器的 addfolder 方法上。Netdisk 控制器的 addfolder 方法代码如下：

```
public function addfolderAction()
{
        $ formData = $ this -> getRequest() -> getParams();
        $ folderid = $ formData[ 'folderid'];
        if (trim( $ formData[ 'foldername']) == ''){
             $ this -> redirect( '/netdisk/list/folderid/'. $ folderid);
             exit();
        }
        $ folder = $ this ->_ufolder -> getUfolder(
        array( 'fid' => $ folderid, 'foldername' => $ formData[ 'foldername']));
        if( $ folder == null){
             $ data = array( 'foldername' => $ formData[ 'foldername'],
                       'layerid' => $ formData[ 'layerid'] + 1,
                       'fid' => $ folderid,
                       'uid' => $ this ->_user -> getIdentity() -> id,
                       'cdat' => time()
             );
             $ newFolder = $ this ->_ufolder -> createDisk( $ data);
             $ this -> redirect( '/netdisk/list/folderid/'. $ folderid);
        }
   }
```

程序首先获取从视图中传递过来的表单参数，接着进行非空与重名检验。如果文件夹名称不为空且当前目录中没有与输入同名的子目录，则调用 Wm_Modele_Ufolder 模型方法 createDisk 创建新的子文件夹。

2. 上传文件

当用户单击视图页面中的“上传文件”按钮后，页面中出现文件上传表单元素，如图 11.4 所示。该表单是一个 Zend_Form 表单对象，由 Common 控制器的 upload 方法视图渲染得到。文件的上传与下载是 Web 应用中的常用操作，许多功能模块都会用到它。为了满足这种通用性的要求，我们将它放在一个专门的公共控制器 Common 中，该控制器的实现与第 8

章中的 8.6 节内容相似,这里不再赘述。

11.4 日程信息管理

办公自动化管理系统最大的优点就是高效与快捷,用户登录系统后,需要能在最短的时间内了解自己的工作任务及其轻重缓急。由此,本系统专门设计了一个固定框架来显示用户的近期工作安排与未完成工作内容,以免用户因遗忘而影响工作。除此之外还设计了一个日程信息管理子模块,包括添加事务、今日工作、本周安排、日程查询与待办提醒 5 项功能,方便用户的查询与管理。系统日程信息管理界面如图 11.5 所示。

11.4.1 创建控制器及方法

为了对日程信息进行管理,需要设置相应的控制器及方法。控制器及方法的创建这里还是采用 Zend Framework 提供的 ZF Tool 工具完成。

1. 创建控制器及方法

启动 Zend Studio 集成开发环境,选择项目工作区中的 wmProject 项目,然后选择 Project 主菜单下的 Zend Tool 菜单项,打开 ZF Tool 工具窗口,输入命令:

```
zf create controller Schedule
```

在默认模块 default 中创建网络空间信息管理控制器 Schedule。然后用同样的方法打开 ZF Tool 工具窗口,分别输入命令:

```
zf create action add Schedule
zf create action work Schedule
zf create action arrange Schedule
zf create action search Schedule
zf create action remind Schedule
```

在控制器 Schedule 中创建 add、work、arrange、search 和 remind 5 个方法,分别用来完成添加日程事务、显示今日工作、输出本周安排、日程查询以及显示待办事务提醒的功能。

2. 初始化控制器

其初始工作与前面的其他控制器相似,代码如下:

```
class ScheduleController extends Zend_Controller_Action
{
  protected $_user = null;
  protected $_schedule = null;
    public function init()
    {
       $auth = Zend_Registry::get('auth');
      if($auth->hasIdentity()){
          $this->_user = $auth;
      }
      $this->_schedule = new Wm_Model_Schedule();
  }
...
}
```

11.4.2 设计数据表模型及方法

为数据表 tb_schedule 创建表模型类，并添加相应的方法代码，操作步骤与前面相同，下面直接给出源代码：

```
class Wm_Model_Schedule extends Zend_Db_Table
{
  protected $_name = 'tb_schedule';
  public function getSchedules($where = array(), $order = null)
  {
        $select = $this->select();
        if (count($where) > 0){
            foreach ($where as $key => $value){
                $select->where($key.' = ?', $value);
            }
        }
        if ($order){
            $select->order($order);
        }
        $result = $this->fetchAll($select);
        if ($result){
            return $result;
        }
        else{
            return null;
        }
  }
  ...
}
```

11.4.3 日程信息管理功能的实现

日程信息管理功能的实现与第 10 章中的事务信息管理基本相似，为了节省篇幅，这里只介绍“日程查询”子模块中的部分功能，其他请参考教材源代码。

如图 11.7 所示，日程查询是通过用户单击页面中的电子台历进行的，当用户单击台历中的某一天时，会在“今日工作”视图页面中显示当天的工作安排。这里的电子台历实现了年、月的前后滚动，使用非常方便。

下面是 Schedule 控制器的 search 方法代码及说明。

```
public function searchAction()
{
        $time = array();
        $date = new Zend_Date();
```

该代码使用 new 运算符创建一个 Zend_Date 对象，用来处理日期与时间。这里类的构造方法中没有带任何参数，表示获取当前系统。

Zend_Date 组件负责 Zend Framework 应用中的日期时间操作，它属于典型的西方式日期解决方案，通过它可以执行日期时间数据的多种运算，在使用前必须设置用户默认时区。

```
$ today = $ date -> toArray();
$ time['year'] = $ today['year'];
$ time['month'] = $ today['month'];
$ time['day'] = $ today['day'];
```

将当前日期的年、月、日从日期时间对象 date 中取出，并存放在 time 数组中。请大家注意从对象中取出数据的方法。

```
$ year = $ this -> getRequest() -> getParam('year');
$ month = $ this -> getRequest() -> getParam('month');
```

接收请求中的“年”、“月”参数，这里处理的是用户单击电子台历中的“年”或“月”的前后滚动按钮操作。

```
if ( $ year){
     $ time['year'] = $ year;
     $ date = $ date -> setYear( $ year);
}
if ( $ month && ( $ month > 0 && $ month < 13)){
     $ time['year'] = $ year;
     $ date = $ date -> setYear( $ year);
     $ time['month'] = $ month;
     $ date = $ date -> setMonth( $ month);
}
```

如果能够接收到请求中的“年”、“月”参数，说明用户正在进行“年”或“月”的滚动操作，此时必须同步更新台历页面，即重新设置日期时间对象 date 中的日期数据。

```
$ days0 = array( $ date -> sub( $ time['day'] - 1,Zend_Date::DAY) -> toArray());
```

对于图 11.7 所示的页面，日历中每月显示的都是 6 周时间，共计 42 天，不属于本月的日期是用暗灰色表示的。代码中的 days0 表示“本月”的第 1 日。这里的 sub 是 Zend_Date 类的方法，其具体用法请参考 Zend Framework 使用手册或源文件。

```
$ days[1] = array( $ date -> sub( $ days0[0]['weekday'],Zend_Date::DAY) -> toArray());
```

获取到“本月”的第 1 日后，再往前推至该周的星期日。这一个“星期日”即是台历显示的第 1 天日期。

```
    for( $ i = 2; $ i < = 42; $ i++){
        $ days[ $ i] = array( $ date -> add(1,Zend_Date::DAY) -> toArray());
    }

    $ this -> view -> days = $ days;
    $ this -> view -> time = $ time;
}
```

得到台历要显示的第 1 天日期后，使用 Zend_Date 类的 add 方法以步长为 1 依次获得台历中的每一天的日期对象，并将其保存在对象数组中。在视图中输出这 42 天，即可看到如图 11.7 所示的效果。

下面是“日程查询”的视图文件代码：

```
<?php require_once 'schedulecss.php'?>
<?php require_once 'scheduleleft.phtml'?>
<div id="message"><div id="mscontent">
<h2  class="pagetitle">日程安排:<b>日程查询 </b></h2>
    <div align="left">
        <ul class="messagemenu">
        <li><a href="/schedule/add">
            <img src="/images/addBtn.jpg" alt="新事务"></a></li>
        <li><a href="#"><img src="/images/queryBtn.jpg" ></a></li>
        </ul>
        <hr />
    </div>
    <div id="msmain">
    <table width="100%" border="0" cellspacing="0" cellpadding="1"
           class=table1 style="font-size:18px;" align="center">
    <tr style="background:#cccccc;text-align:ceter;">
     <td colspan="2" style="color:#ff6600;height: 30x;text-align:center">
     <a href="javascript:window.location.href='<?php echo $this->baseUrl
             ('/schedule/search/year/'.($this->time['year']-1)) ?>'">
        <img src='/images/prev.gif' /></a>  年   
     <a href="javascript:window.location.href='<?php echo $this->baseUrl
             ('/schedule/search/year/'.($this->time['year']+1)) ?>'">
        <img src='/images/next.gif' /></a>
     </td>
     <td colspan="3" align="center"><?php echo $this->time['year'] ?>
          年<?php echo $this->time['month'] ?>月</td>
     <td colspan="2" align="center">
    <a href="javascript:window.location.href='<?php echo $this->baseUrl
             ('/schedule/search/year/'.$this->time['year'].'/month/'.
             ($this->time['month']-1)) ?>'">
        <img style="cursor:hand" src='/images/prev.gif' />
    </a>  月   
    <a href="javascript:window.location.href='<?php echo $this->baseUrl
            ('/schedule/search/year/'.$this->time['year'].'/month/'.
            ($this->time['month']+1)) ?>'">
        <img style="cursor:hand" src='/images/next.gif' /></a>
     </td>
</tr>
<tr>
     <td align=center class=tr_head width=14% height=40>
          <font color="red">日</font></td>
     <td align=center class=tr_head width=14%>一</td>
     <td align=center class=tr_head width=14%>二</td>
     <td align=center class=tr_head width=14%>三</td>
     <td align=center class=tr_head width=14%>四</td>
     <td align=center class=tr_head width=14%>五</td>
     <td align=center class=tr_head width=14%>六</td>
</tr>
```

```
<?php foreach ( $ this -> days as $ day ){
  if( $ day[0][ 'weekday'] == '7')
      echo "< tr >";
      echo "< td align = center style = 'cursor:hand;line - height:30px;";
      if ( $ day[0][ 'month']!= $ this -> time[ 'month'])
          echo "color: # cccccc;";
      if ( $ day[0][ 'weekday'] == '7' &&
                              $ day[0][ 'month'] == $ this -> time[ 'month'])
          echo "color: # ff0000;";
      if ( $ day[0][ 'day'] == $ this -> time[ 'day'] &&
                              $ day[0][ 'month'] == $ this -> time[ 'month'])
        echo "background: # ff8800;";
        echo "'"." onclick = javascript:window. location. href =
            "."'/schedule/work/year/". $ day[0][ 'year']."/month/
            ". $ day[0][ 'month']."/day/". $ day[0][ 'day']."'>";
            echo $ day[0][ 'day']."</td>";
    if( $ day[0][ 'weekday'] == '6')
        echo "</tr>";
}?>
</table></div></div></div>
```

注意：上述代码中的 6 个阴影部分，它们分别实现事务的添加、年份的前后滚动、月份的前后滚动、不同日期的背景颜色设置以及用户单击某日后的页面跳转功能。

11.5 本章小结

本章介绍了办公自动化管理系统中常用办公功能的实现，包括用户网络空间、日程安排的创建与管理。用户网络空间类似于一个简单的资源管理器，能够进行文件夹的创建与删除、文件的复制与粘贴等基本功能。

本章在实现应用功能的过程中进一步介绍了 Zend Framework 视图助手的使用方法，同时引入了新的 Zend_Date 时间日期组件，这些内容是本章的学习重点。

第 12 章　用户权限及系统优化

通过前面章节的介绍，一个办公自动化管理系统的 Web 应用已经基本成形。但细心的读者可能会发现，在整个系统中，除了在第 7 章的用户认证中写过少量的访问控制代码外，没有编写其他任何关于用户权限的内容。如果任意一个访问者都可以访问像系统后台管理中心这样的功能模块，那么这个后台管理就没有什么作用和意义了。系统除了用户访问控制权限方面的缺憾之外，在其他许多地方还存在一些需要完善的地方。

本章实现办公自动化管理系统的访问控制、缓存等 Web 应用优化技术方案，由此介绍 Zend Framework 的 Zend_Acl 以及 Zend_Cache 组件。另外，还将通过介绍验证码、系统日志以及数据备份的实现对系统进行简要完善。

12.1　Zend_Acl 访问控制

Zend Framework 中的访问控制通过 ACL 访问控制列表(Access Control List)来实现。所谓访问控制列表，通俗地讲就是权限控制，应用程序可以利用这样的功能限制某一类用户来访问特定保护的对象。其中，用户常被称为角色(Role)，被访问的对象被称为资源(Resource)，所以 ACL 的核心功能就是允许或限制某一类 Role 访问某些 Resource。在我们的应用中，ACL 中的角色即用户的身份，资源即系统内容，而权限就是角色能访问哪些内容，不能访问哪些内容。

使用 Zend Framework 中的 Zend_Acl 组件即可实现完整的访问控制。该组件提供了一整套实现访问控制列表的解决方案。Zend_Acl 组件的权限控制设计得非常完美，理解和使用简单，通用性也非常好。

12.1.1　资源与角色

Zend Framework 的 Zend_Acl 组件中定义了两个重要的概念，即资源与角色。资源是指一个被限制访问的对象；角色则是指可以发出请求来访问资源的对象。Zend_Acl 组件中分别以 Zend_Acl_Role 类与 Zend_Acl_Resource 类表示资源与角色。

1. 资源

在 Zend Framework 中，resource 可以是任何事物，例如一个 Controller、一个 Action、一个 Model 或者一个 File 等。可以简单地把资源理解为一个可供访问的对象，例如某一个页面。

Zend_Acl 组件提供了一个树结构，它可以添加多个 Resource(或者叫"访问控制下的区域")。因为 Resource 被存储在这样一个树结构中，所以它们可以被组织成从一般(树根)到

特殊(树叶)。基于特殊 resource 的查询将自动从 Resource 的等级结构中搜索每个资源的父类 Resource 所具有的规则,它允许简单的规则继承。例如,要把一个默认的规则应用到系统中的功能模块的每个页面,就简单地把这个规则分配给该模块的控制器,而不是把规则分配给该控制器的每个页面。然而,有些页面也许要求特定的规则,这在 Zend_Acl 里也是很容易实现的。一个 Resource 可以从唯一的一个父 Resource 继承,而这个父 Resource 也有它自己的父 Resource 等。

在 Zend_Acl 中,创建一个资源非常简单。Zend_Acl 提供了 Zend_Acl_Resource_Interface 接口,该接口使开发者可以非常方便地创建 Resource,为了使 Zend_Acl 把某个对象当作一个 Resource,一个类只需要实现包含了方法 getResourceId()的接口即可。另外,Zend_Acl_Resource 是一个包含在 Zend_Acl 里作为一个基本的 Resource 实现的类,开发者可以任意对其进行扩展。

2. 角色

与资源相对应,在 Zend_Acl 组件中,角色就是一个发出希望访问某个资源请求的对象,通常理解为一个用户所具有的身份,这个身份由我们自行定义。在 Zend_Acl 中,Role 支持规则的继承,一个 Role 可以从一个或多个 Role 继承。例如在本系统中,我们设置普通职工的角色为 user、部门管理员的角色为 editor,角色 editor 继承于角色 user,它拥有 user 的所有访问权限。

角色与资源一样,其创建过程也非常简单。Zend_Acl 提供了 Zend_Acl_Role_Interface 接口,为了让 Zend_Acl 把某个对象当作一个 Role,一个类只需要实现这个只包含了一个 getRoleId()方法的接口即可。与 Zend_Acl_Resource 一样,Zend_Acl_Role 也是一个包含在 Zend_Acl 中作为一个基本的 Role 实现的类。

12.1.2 Zend_Acl 的创建与使用

在理解了 Zend_Acl 中的资源与角色后,下面介绍如何创建并使用访问控制列表。

1. 创建 ACL

从上面关于访问控制列表、资源与角色的定义可以看出,ACL 可以表示任何一组物理或虚拟对象。创建一个新的 ACL 对象,可以使用 new 关键字不带参数地实例化 Zend_Acl 类。

```
$ acl = new Zend_Acl();
```

注意:如果不定义任何访问权限,则默认的 ACL 将禁止所有的 Role 访问任何 Resource。

2. 添加资源

将资源添加到访问控制列表中可以使用 Zend_Acl 实例的 addResource 方法,该方法的语法格式如下:

```
addResource( $ resource, $ parent = null)
```

其中,参数 resource 为需要添加的资源,可以为字符串、数组或者源源对象;可选参数 parent 为参数 role 继承的父资源。

3. 注册角色

要将指定的角色注册到访问控制列表中,可以使用访问控制列表实例的 addRole 方法。

该方法的语法格式如下：

```
addRole( $ role, $ parents = null)
```

其中，参数 role 为需要注册的角色，可以为角色对象、字符串或者数组；可选参数 parents 为参数 role 继承的父角色。

4. 定义访问控制

在访问控制列表中，为指定注册角色添加允许权限可以使用 ACL 对象的 allow 方法。该方法的语法格式如下：

```
allow ( $ roles = null, $ resources = null, $ privileges = null, Zend_Acl_Assert_Interface
 $ assert = null)
```

其中，参数 role 为指定的角色，可以为角色、字符串或者数组；参数 resource 为允许使用的资源，可以为资源、字符串或者数组，默认值为 null，即所有资源；参数 privileges 为允许使用的权限，可以为字符串或者数组；参数 assert 为实现了 Zend_Acl_Assert_Interface 接口对象，用来定义访问的条件。

相应地，也可以为角色添加拒绝权限，使用 ACL 对象的 deny 方法即可为角色添加拒绝权限。该方法的语法格式如下：

```
deny( $ role, $ resource, $ privilege, $ assert);
```

其中各参数的意义与方法 allow 相同。执行该方法将为指定角色添加拒绝权限。

5. 移除控制规则

由于某种需要，在使用访问控制列表时可能需要对其中定义的控制规则进行移除，此时可以使用 ACL 对象的 removeAllow 方法与 removeDeny 方法将指定规则进行移除。这两个方法的语法格式分别如下：

```
removeAllow( $ roles = null, $ resources = null, $ privileges = null)
removeDeny( $ roles = null, $ resources = null, $ privileges = null)
```

其中各参数的意义与方法 allow 相同。它们所带的参数均可以省略，当不带任何参数执行方法时，表示移除所有用户资源的所有允许或拒绝操作。

6. 查询控制状态

使用访问控制列表对象的 isAllowed 方法可以对指定角色是否有指定权限进行查询，执行该方法将会根据指定角色是否有某种操作权限返回相应的布尔值。其语法格式如下：

```
isAllowed( $ role = null, $ resource = null, $ privilege = null)
```

其中各参数的意义与方法 allow 相同。

12.2 系统访问控制的实现

Zend Framework 提供的 Zend_Acl 组件功能强大、使用灵活且直观。在实际应用过程中，我们需要将 ACL 文件写成插件形式，并在其中定义资源和角色，设置好相应的规则，这样它就可以很好地工作。下面实现本系统案例项目的访问控制功能。

12.2.1 系统角色及权限的设置

在第 6 章中创建了用户信息数据表 tb_user，其中有一个名为 role 的字段，它表示的就是用户的角色信息。为了简单，本系统设置 4 种角色，即 admin、editor、user 和 guest，它们分别表示"系统管理员"、"部门管理员"、"普通用户"及"匿名用户"。

匿名用户拥有最小的权限，只能访问系统封面主页、用户登录及系统功能与使用说明、重置密码页面，也就是说没有登录的用户不能进入系统内部；系统管理员用户享有全部权限，可以访问所有资源。介于这两者之间的是普通用户与部门管理员，其中普通用户在系统前台管理中有较多权限，可以浏览、创建、更新甚至删除某些内容，但不能进入后台管理系统；而部门管理员拥有系统管理员的部分权限，可以通过后台管理中心添加、编辑、删除部分内容。例如，可以添加单个用户注册信息，但不能批量注入。

从 guest 到 user，再到 editor、admin，越往后权限越大，前者拥有的权限后者都有，后者拥有的权限前者不一定拥有，即后者的权限包含前者，这就是前面所讲的角色的继承。

通过第 2 章的学习我们已经知道，Zend Framework 应用的资源内容是通过 MVC 形式被创造和访问的，其中的 C(也就是控制器及其方法)是调度这些内容的核心。当我们罗列出控制器及其方法时，也就列出了应用的资源。然后指定相应角色的权限，即规定哪些角色可以访问哪些资源，哪些角色不可以访问哪些资源，这样就可以实现整个系统的访问控制了。

表 12.1 所示为本系统中资源、角色及权限的对应关系。由于系统资源较多，表中只列出了部分内容，其他内容与表中内容类似。

表 12.1 系统资源、角色及权限对应表

资源			角色			
控制器	模块	方法	guest	user	editor	admin
index	default	index	√	√	√	√
		main	×	√	√	√
	admin	index	×	×	√	√
user	default	index	×	√	√	√
		login	√	√	√	√
		logout	×	√	√	√
		register	×	√	√	√
		account	×	√	√	√
		resetpassword	√	√	√	√
		update	×	√	√	√
	admin	index	×	×	√	√
		insert	×	×	×	√
		register	×	×	√	√
		delete	×	×	×	√
		search	×	×	√	√
		list	×	×	√	√
		detail	×	×	√	√

续表

资源			角色			
控制器	模块	方法	guest	user	editor	admin
document	default	index	×	√	√	√
		list	×	√	√	√
		docdetail	×	√	√	√
		issue	×	×	√	√
		receive	×	√	√	√
	admin	index	×	×	√	√
		insert	×	×	×	√
		register	×	×	√	√
		delete	×	×	×	√
		search	×	×	√	√
		list	×	×	√	√
		detail	×	×	√	√
affair	default	index	×	√	√	√
		add	×	√	√	√
		list	×	√	√	√
		reply	×	×	√	√
	admin	index	×	×	√	√
		list	×	×	√	√
		delete	×	×	×	√
…	…	…	…	…	…	…

设置好整个系统的权限后，接下来就可以编写代码实现系统的访问控制了。在编写代码之前还必须弄清楚一个问题，就是访问控制代码文件应该在什么时候被执行。判断用户能否访问某一资源并做出相应的控制措施需要在框架处理具体的控制器方法之前触发，所以访问控制的实现需要以插件的形式完成。

12.2.2 开发系统 ACL 插件

使用 Zend_Acl 组件实现访问控制分为以下 3 个步骤：

(1) 继承 Zend_Controller_Plugin_Abstract 类建立 plugin 插件，通过 preDispatch()方法定义插件什么时候执行；

(2) 建立角色和资源；

(3) 对每个资源加入角色权限控制。

1. 编写插件

首先在项目的 library 目录下创建 wmoams\controllers\plugins 目录，然后在其中创建 Acl.php 插件文件，这个插件只需要包含并扩展抽象类 Zend_Controller_Plugin_Abstract，代码如下：

```
class WmOAMS_Controller_Plugin_Acl extends Zend_Controller_Plugin_Abstract
{
    public function preDispatch(
                                Zend_Controller_Request_Abstract $request)
```

```
    {
        $ acl = new Zend_Acl();
        $ acl -> addRole('guest');
        $ acl -> addRole('user', 'guest');
        $ acl -> addRole('editor', 'user');
        $ acl -> addRole('admin');
        $ acl -> addResource('index');
        $ acl -> addResource('user');
        $ acl -> addResource('admin. index');
        …
        $ acl -> addResource('news');
        $ acl -> addResource('schedule');
        $ acl -> deny('guest',null,null);
        $ acl -> allow('guest', 'index',array('index'));
        $ acl -> allow('guest', 'user',array('login', 'resetpassword'));
        $ acl -> allow('user', 'index',array('main'));
        $ acl -> allow('user', 'user',array('index', 'logout', 'register', 'account', 'update'));
        …
        $ acl -> allow('editor', 'admin. index',array('index'));
        $ acl -> allow('editor', 'admin. user',array('index', 'register', 'list', 'search', 'detail'));
         …
$ acl -> allow('admin',null,null);
        $ auth = Zend_Auth::getInstance();
        if( $ auth -> hasIdentity()) {
            $ identity = $ auth -> getIdentity();
            $ role = strtolower( $ identity -> role);
        }else{
            $ role = 'guest';
        }
        $ module = $ request -> module;
        $ controller = $ request -> controller;
        $ action = $ request -> action;
        if( $ module === 'default'){
            $ resource = $ controller;
        }else{
            $ resource = $ module. '. '. $ controller;
        }
        $ isExist = array_search( $ resource, $ acl -> getResources());
        if(! $ isExist && $ isExist !== 0 ){
            if ( $ role == 'guest' ) {
                $ request -> setControllerName('index');
                $ request -> setActionName('index');
            } else {
                $ request -> setControllerName('error');
                $ request -> setActionName('noresource');
            }
        }elseif(! $ acl -> isAllowed( $ role, $ resource, $ action)) {
                if ( $ role == 'guest' ) {
                    $ request -> setControllerName('index');
                    $ request -> setActionName('index');s
                } else {
                    $ request -> setControllerName('error');
                    $ request -> setActionName('noac');
                }
            }
        }
}
```

在讲解上述代码之前先来介绍一下插件。所谓 plugin 插件，顾名思义就是要在正常的框架执行流程中的某个环节处打断原有的流程，插入由开发者自行开发的插件程序。一个 plugin 必须继承 Zend_Controller_Plugin_Abstract 类并重载某些方法来实现相应的功能，其中常用的方法有下面几种。

- routeStartup：在 Zend_Controller_Front 向注册的路由器发送请求前被调用。
- routeShutdown：在路由器完成请求的路由后被调用。
- dispatchLoopStartup：在 Zend_Controller_Front 进入其分发循环前被调用。
- preDispatch：在动作由分发器分发前被调用。该回调方法允许代理或者过滤行为，通过修改请求和重设分发标志位使当前动作可以跳过或者被替换。
- postDispatch：在动作由分发器分发后被调用。该回调方法允许代理或者过滤行为，通过修改请求和重设分发标志位指定新动作进行分发。
- dispatchLoopShutdown：在 Zend_Controller_Front 退出其分发循环后调用。

preDispatch 在调用具体 action 之前触发，所以在这个地方可以添加权限认证、URL 转发等动作；而 postDispatch 在调用 action 之后触发，在这个地方可以做输出缓存等。

关于路由的问题，对初学者来说有一定的难度，本书不做介绍。有了上面关于插件的一些基本知识以后，我们再来看看上面的插件代码。

代码共分为下面 8 个部分：

1）插件类的定义

基类为 Zend_Controller_Plugin_Abstract 类的 WmOAMS_Controller_Plugin_Acl 类中实现了一个名为 preDispatch 的方法，该方法的参数是当前的一个 request 请求，如上所述，这个方法被定义为将在动作由分发器分发前被调用，也就是说访问控制代码是在控制器方法 action 被分发前执行的。如果在执行访问控制代码时发现发出请求的用户没有访问该控制器方法的权限，那么原来的请求路径就会被更改，这样该用户的此次请求就不能访问到请求的视图页面，从而实现了对访问的控制。

2）创建 ACL 对象

通过 new Zend_Acl()的形式创建名为 acl 的 ACL 对象。

3）注册角色

使用 ACL 对象的 addRole 方法注册系统角色。其中，user、editor 采用了继承的方式；admin 拥有全部权限，不需要继承。

4）添加资源

使用 ACL 对象的 addResource 方法添加系统资源。如上所述，这里的资源指系统中的控制器。由于系统模块中的控制器可能重名，我们把资源名前加上了控制器所属模块的名字，如果是默认模块，则不添加。例如，代码中的资源 index、admin. index 分别代表默认模块 default 和后台管理模块 admin 中的 index 控制器。

5）定义访问控制

使用 ACL 对象的 deny 和 allow 方法定义角色的访问控制，访问控制规则遵循表 12.1 中的设定。deny 和 allow 方法中的第 3 个参数表示访问权限，我们用控制器的方法名来设置。也就是说，只要能在访问权限字符中找到相应控制器的方法即可访问该方法对应的视图页面。

6）获取用户角色

根据用户登录认证信息获取用户角色信息，若用户未登录，则为匿名用户 guest。

7）获取资源并判断是否存在

从请求对象 request 中获取模块、控制器及方法。若为默认模块，直接将控制器名当作资源；若为其他模块，则在控制器名前加“模块名.”后赋给资源变量 resource。

资源的存在与否通过 PHP 的函数 array_search 的返回值判断。若资源存在，返回注册资源的索引，注意有索引为 0 的资源；若资源不存在，返回 false。从上面的代码可以看出，若匿名用户访问本系统，请求资源不存在时，页面一律跳转至系统封面页面；若是登录用户请求的资源不存在，则统一跳转到错误控制器 Error 的“资源不存在”提示页面。

8）检验访问权限

使用 ACL 对象的 isAllowed 方法检验用户的权限。若匿名用户没有访问权限，则页面跳转至系统封面；若登录用户没有访问权限，则页面跳转到错误控制器的“资源不存在”提示页面。

2. 注册插件

访问控制插件代码编写完成之后，还需要在配置文件中添加代码开启它。打开配置文件 application\cofigs\application.ini，在 production 小节添加如下代码：

```
autoloaderNamespaces[] = "WmOAMS_"
resources.frontController.plugins.acl = "WmOAMS_Controller_Plugin_Acl"
```

到此为止，一个功能强大、易于维护和扩展的权限控制系统就完成了。整个系统已经置于 ACL 的控制之下，访问系统中的任何一个页面都要经过插件中 ACL 规则的检查。只有符合 ACL 访问规则的页面才会正常呈现在访问者的面前，否则系统会重新定向到系统封面、错误提示页面或者抛出错误。

12.3 系统的优化

Web 应用系统的性能受到各种因素的影响，因此系统的优化必须考虑到整个系统的方方面面，侧重点不同、应用场景不同，采用的优化策略也应该不尽相同。评价 Web 应用系统性能主要有两项指标，一是客户端响应时间，二是服务器的吞吐量，因此系统的优化应围绕这两个方面展开。另外，Web 应用系统的性能优化还必须考虑系统的开发语言与采用的设计模式。本节主要介绍数据缓存的性能优化策略。

12.3.1 Zend_Cache 数据缓存

数据缓存作为一种 Web 应用的性能优化方法，是 Web 应用程序几乎都要用到的技术方案。对于访问量较大的应用来说，使用缓存可以极大地减轻数据库服务器的压力，解决由此产生的性能瓶颈。

Zend Framework 的 Zend_Cache 是一个理想的缓存组件，提供了缓存程序中任何数据的方法。下面简单介绍缓存原理及 Zend_Cache 的使用。

1. 缓存实现原理

缓存是指将一些数据存放在固定的载体（如 Session、Cookie、文件或者数据库）中，需要

时再读出的这样一个过程。在 Zend_Cache 缓存架构中，数据的缓存由前端（Frontend）和后端（Backend）相互配合完成。

Zend_Cache 中的前端与后端互有分工，前端提供缓存的对象，通过后端适配器和一个灵活的 IDs 与 Tags 系统存储缓存记录，以使访问精准定位到前端缓存页面。数据缓存一般按照以下流程实现：

- 从缓存文件夹中寻找是否有符合条件的缓存文件；
- 如果找到，即有符合条件的缓存文件，而且没有过期，直接将该文件传递给浏览器；
- 如果没找到，即没找到符合条件的缓存文件，则检查文件是否已过期；
- 如果没找到，创建该文件，存入缓存文件夹，同时将文件输出到浏览器。

2. Zend_Cache 前端

Zend_Cache 中的前端在整个数据缓存过程中起缓存操作的作用，它实际上是一些预定义类，主要有 Zend_Cache_Core、Zend_Cache_Frontend_Output、Zend_Cache_Frontend_Function、Zend_Cache_Frontend_Class、Zend_Cache_Frontend_File、Zend_Cache_Frontend_Capture 以及 Zend_Cache_Frontend_Page 等，其中，Zend_Cache_Core 类是缓存前端的核心类，也是其他前端类的基类。

3. Zend_Cache 后端

Zend_Cache 中的后端负责对缓存数据进行保存，主要包括 Zend_Cache_Backend_File、Zend_Cache_Backend_Sqlite、Zend_Cache_Backend_Memcached、Zend_Cache_Backend_Apc、Zend_Cache_Backend_Xcache、Zend_Cache_Backend_ZendPlatform 等，常用的是 Zend_Cache_Backend_File 后端，它把缓存数据存储到一个指定的目录文件中。

4. 创建 Zend_Cache 实例

缓存对象的创建通过 Zend_Cache 类的静态方法 factory 实现，该方法的语法格式如下：

```
factory( $ frontend, $ backend, $ frontendOptions = array(), $ backendOptions = array())
```

其中，参数 frontend 为调用的 Zend_Cache 前端组件；参数 backend 为调用的 Zend_Cache 后端组件；参数 frontendOptions 与参数 backendOptions 分别为前端与后端的设置数组。在以上 4 个参数中，后两个参数为可选项，如果省略设置参数，方法将采用其默认值。

5. 向缓存中写入数据

向缓存中写入数据使用 Zend_Cache_Core 的 save 方法实现，其语法格式如下：

```
save( $ data, $ id = null, $ tags = array(), $ specificLifetime = false)
```

其中，参数 data 为需要放入缓存中的数据内容，其类型为 mixed 混合型；可选参数 id 为缓存的 ID 号，如果省略该值，系统将使用最后的缓存 ID；参数 tags 为缓存的标识；可选参数 specificLifetime 为一个布尔型变量，该参数指定是否为缓存数据设置一个特殊的生命期，其默认值为 false。执行该方法将会把缓存数据写入到指定的前端缓存中。

6. 清除缓存数据

当缓存数据不再需要时，出于安全性及节省系统资源的考虑，应该将其清除。该操作通过 Zend_Cache_Core 的 remove 方法以及 clean 方法实现，其中 remove 方法用于清除单个缓存记录。其语法格式如下：

```
remove( $ id)
```

其中，参数 id 为指定的缓存 ID 号。

clean 方法通常用于清除多个缓存记录，它的语法格式如下：

```
clean( $ mode = 'all', $ tags = array())
```

其中，参数 mode 为清除模式，该参数的可选值如下。

- all：清除所有缓存，该值为默认值；
- old：仅清除过期的缓存；
- matchingTag：清除与参数 tags 指定的标记内容相匹配的所有缓存；
- notMatchingTag：清除与参数 tags 指定的标记内容不相匹配的所有缓存。

参数 tags 是参数 mode 为 matchingTag 或者 notMatchingTag 时指定的标记类型，该参数可以是数组、字符串或者字符。

12.3.2 Zend_Cache 数据缓存实例

在了解了 Zend_Cache 的数据缓存原理之后，下面分析一个缓存实例。该实例通过缓存获取本书案例项目中的所有部门信息。

在第 4 章的 4.1 节中已经创建了一个名为 Cache 的缓存对象，并把它存放在了注册表中，这里只需要将其取出使用即可。

1. 缓存参数

缓存的前端与后端参数可以在创建时直接通过数组设置，也可以通过配置文件 application.ini 配置。启动 Zend Studio 集成开发环境，打开项目 wmProject 的配置文件 application\cofigs\application.ini，可以看到如下代码：

```
cache.leftTime = "7200"
cache.cache_dir = APPLICATION_PATH "/tmp/"
```

代码中的第 1 行设置缓存的生命期，这里为两个小时，如果该值为空，则缓存永久有效。代码的第 2 行设置缓存文件存储目录。

2. 缓存对象

打开 application\Boostrap.php 引导文件，在初始化函数 initRegister 中可以看到如下代码：

```
$ frontOptions = array('leftTime' => $ config['cache']['leftTime'],
                       'automatic_serialization' => true);
$ backOptions = array('cache_dir' => $ config['cache']['cache_dir']);
$ cache = Zend_Cache::factory('Core', 'File',
                              $ frontOptions, $ backOptions);
Zend_Registry::set('cache', $ cache);
```

代码的第 1 句设置缓存前端参数，automatic_serialization 表示缓存是否可自动序列化，序列化后可存储非字符数据；代码的第 2 句设置后端存储目录；代码的第 3 句创建缓存对象，缓存前端采用的是 Core，也就是 Zend_Cache_Core 类，后端采用的是 File，也就是 Zend_Cache_Backend_File 类。该缓存对象用文件存储缓存数据，并将文件存放到 application\

tmp 目录中。

3. 缓存对象的使用

在 Common 控制器中创建 test 方法演示缓存的使用，下面是 test 方法中的代码：

```
public function testAction()
{
        $cache = Zend_Registry::get('cache');
        $departCache = $cache->load('depart');
        if(!$departCache){
            $departModel = new Wm_Model_Depart();
            $depart = $departModel->getDeparts();
            $cache->save($depart,'depart');
        }else {
            echo '<pre>';
            print_r($departCache);
            echo '</pre>';
        }
    }
```

程序首先从注册表中获取缓存对象，然后调用 Zend_Cache_Core 类的 load 方法加载指定的缓存数据；如果指定的缓存不存在，则通过调用 Zend_Cache_Core 类的 save 方法创建它，否则直接使用即可。

4. 缓存使用效果

在浏览器的地址栏中输入 http://wmoams.com/common/test，第 1 次渲染页面，创建 depart 缓存，刷新页面，效果如图 12.1 所示。



图 12.1　缓存示例效果

12.4 动作助手和系统缓存的实现

12.3 节介绍了 Zend_Cache 缓存的原理及使用方法，本节实现系统的缓存功能，这里主要实现系统的页面缓存。Zend Framework 应用中的页面缓存可以通过 12.3 节介绍的方法实现，还可以通过“动作助手”实现。

12.4.1 Zend Framework 动作助手

通过 Zend Framework 提供的助手模式可以把一些经常使用的功能模块进行封装，这样就可以在需要的地方灵活使用。Zend Framework 中有两种助手，即动作助手与视图助手，对于视图助手已经在第 11 章做了详细的介绍。

Zend Framework 的动作助手可以向任何 Zend_Controller_Action 的派生动作控制器中即时加入功能，以使增加公共的动作控制器功能时尽量减少派生动作控制器类的创建。动作助手在需要调用时加载，可以在请求的时候或者动作控制器创建的时候实例化。常见的动作助手如下：

- FlashMessenger：用来处理 Flash Messenger 会话；
- Json：用来解码和发送 JSON 响应；
- Url：用于创建 Urls；
- Redirector：用于将程序重定向到内部或者外部页面；
- ViewRenderer：自动完成在控制器内建立视图对象并渲染视图的过程；
- AutoComplete：自动响应 AJAX 的自动完成；
- ContextSwitch、AjaxContext：为动作提供替代响应格式；
- Cache：实现缓存的相关操作；
- ActionStack：用于操作动作堆栈。

这些动作助手存放在 Zend Framework 库文件的 Zend\Controller\Action\目录中，如图 12.2 所示。

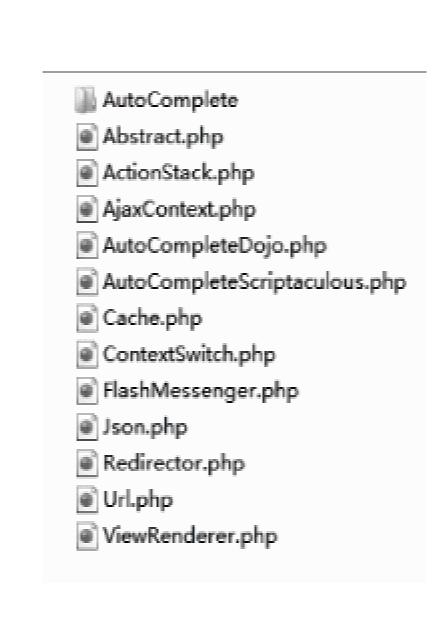


图 12.2　常见动作助手

12.4.2 系统缓存的实现

1. 配置缓存

打开 application\cofigs\application.php 配置文件，添加如下代码：

```
resources.cachemanager.page.backend.options.public_dir = APPLICATION_PATH "/../public/cache"
resources.cachemanager.pagetag.backend.options.cache_dir = APPLICATION_PATH "/../public/cache/data/tags"
resources.frontController.params.disableOutputBuffering = true
```

以上配置定义了两个目录，public_dir 用来存放前端生成的缓存文件，cache_dir 用来存放后端的 Tags 标识。最后一行用于关闭程序预设的 OutputBuffer 功能，以避免缓存失效。配置的缓存目录应确保已经存在，并具有写入权限。

2. 使用缓存

经过以上配置后就可以简单使用缓存了，只需要在控制器的 init 初始化方法中使用动作助手指定要缓存的页面，并为其指定一个标识即可。

例如，在 Index、User、Document 控制器的 init 方法中分别添加如下代码：

```
$ this->_helper->cache(array('index','main'),array('defaultindex'));
$ this->_helper->cache(array('index','login'),array('defaultuser'));
$ this->_helper->cache(array('index','list','receive'),
        array('defaultdocument'));
```

即可实现这些控制器相应页面的缓存。代码中的 cache 动作助手带两个参数，前一个是要存储的缓存页面，后一个是为这些页面指定的 Tags 标识。

在浏览器中访问上述缓存的控制器的页面后，缓存目录中的内容如图 12.3 所示。

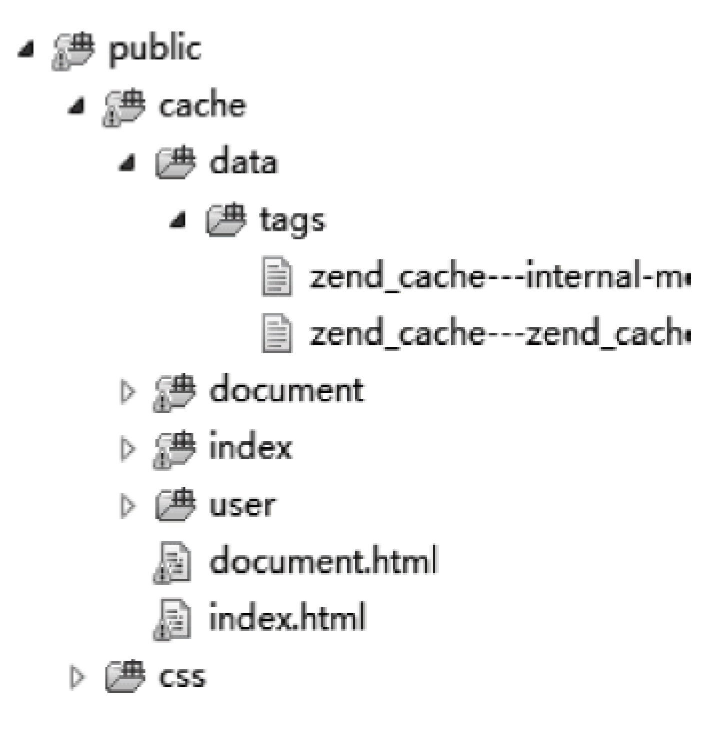


图 12.3　缓存目录内容

从图 12.3 可以看出，在浏览器访问了相应缓存页面后，缓存目录中生成了一些目录、HTML 缓存页面和一些没有扩展名的标记文记，并且这些内容仍然是按照框架的层次结构存储的，显得井然有序。

页面被缓存之后，如果有下一个请求再访问该页面，框架会首先到 public\cache 目录下按照 URL 的组织方式查找对应的 HTML 文件；如果找到，直接输出该 HTML；如果没找到，交给框架，按照原来的路由流程进行处理。

12.5　系统的完善

通过前面章节的介绍，本书案例项目——微梦办公自动化管理系统——的基本功能实现完毕，其他功能模块的实现与前面讲的大同小异，希望大家在此基础上继续完成余下的工作。

由于本书重点介绍 Zend Framework 框架及组件的使用，所以在系统设计中采用的是一些最基础的、最简单的业务处理逻辑。这样，程序代码可能不太严谨，甚至有些冗余，业务处理逻辑有时可能会显得比较笨拙，希望大家在学习过程中结合自己的实际情况对页面布局、数据库设计以及业务处理逻辑进行适当的完善。

本节介绍 Web 应用中的图形验证码、系统日志及数据备份的实现。

12.5.1 验证码

为了防止非法用户通过恶意程序采用试探密码的方式登录本系统，或连续向数据表中注入垃圾信息，应该在系统登录、发布公文、添加消息、添加事务等操作过程中添加验证码信息。

在 Web 应用中，验证码可以有多种形式，例如文字验证码、图片验证码、语言验证码等，下面采用 Web 应用中最常用的验证码形式（即图片验证码），来实现系统后台管理中心的登录验证功能。为了简单，这里只实现验证码功能，效果如图 12.4 所示。



图 12.4　后台管理中心登录验证

本系统后台管理中心的登录验证是在前台页面中进行的。当用户通过认证进入系统后，如果用户具有管理权限，则会在页面的顶部显示“进入系统后台管理中心”超级链接，单击该链接，显示如图 12.4 所示的验证页面。

1. 编写验证码类

使用 PHP 语言制作图片验证码可以使用 GD2 函数。考虑到验证码的高度、宽度以及图片中文字的字体在不同类型的表单中可能存在差异，这里单独将验证码的实现过程封装成验证码类。

打开 Zend Studio 集成开发环境，在项目 wmProject 中添加新的子目录 application\plugins\util，并添加名为 ValidateCode.php 的文件，在文件中创建一个名为 Wm_Plugin_Util_ValidateCode 的自定义类。其代码如下：

```
class Wm_Plugin_Util_ValidateCode
{
    private $_width;                                    //验证码图片宽度
    private $_height;                                   //验证码图片高度
    private $_codeStr;                                  //验证码
    private $_fontType;                                 //验证码字体类型
    private $_img;                                      //验证码图像句柄

    public function __construct ($width, $height, $codeStr = '0000', $fontType = 0)
    {
        $this->_width = $width;
        $this->_height = $height;
        $this->_codeStr = substr($codeStr,0,4);
        $this->_fontType = $fontType;
    }
    private function _getColor ($x, $y)                 //获取颜色
    {
```

```
            $ r = mt_rand( $ x, $ y);
            $ g = mt_rand( $ x, $ y);
            $ b = mt_rand( $ x, $ y);
            return imagecolorallocate( $ this->_img, $ r, $ g, $ b);     //返回颜色句柄
        }
        private function _init ()                                        //创建初始图像
        {
            $ this->_img = imagecreate( $ this->_width, $ this->_height);
        }
        private function _build ()                                       //创建验证码
        {
            //为图像填充背景色
            imagefill( $ this->_img,0,0, $ this->_getColor(150,250));
            //创建一个矩形作为验证码的边框
            imagerectangle( $ this->_img,0,0, $ this->_width - 1,
                          $ this->_height - 1, $ this->_getColor(50,150));
            if ( $ this->_fontType == 0) {
                $ fontFileName = 'ARIALBI.TTF';                          //设置验证码文字的字体
            } else {
                $ fontFileName = 'ARIALN.TTF';
            }
            //绘制文字
            for ( $ i = 0;  $ i < strlen( $ this->_codeStr);  $ i ++) {
                imagettftext( $ this->_img,mt_rand(12,24),0,
                    ( $ this->_width) / 4 *  $ i,mt_rand(20, $ this->_height - 5),
                    $ this->_getColor(10,180),APPLICATION_PATH .
                                      '/resources/font/' .  $ fontFileName,
                    substr( $ this->_codeStr, $ i,1));
            }
            //绘制 15 条干扰线
            for ( $ i = 0;  $ i < 15;  $ i ++) {
                imageline( $ this->_img,mt_rand(0, $ this->_width),
                    mt_rand(0, $ this->_height),mt_rand(0, $ this->_width),
                    mt_rand(0, $ this->_height), $ this->_getColor(110,210));
            }
        }
        //显示图像
        public function show ()
        {
            header('content-type:image/png');                            //设置输出图片的格式
            $ this->_init();
            $ this->_build();
            imagepng( $ this->_img);                                     //输出图片
        }
    }
```

上述代码使用的都是 PHP 函数,比较简单,请读者结合注释自行理解,这里要注意验证码类的名字与其存放的目录及文件名之间的联系。在第 2 章中已经介绍过,Zend Framework 框架具有自动加载的功能,如果自定义类符合特定的规则,在程序中使用这些

类时就不需要使用像 include 这样的包含语句。

另外，验证码字体存放在项目的 application\resources\font 目录中，请大家参考教材源代码。

2. 封装验证码表单元素

由于验证码一般和表单元素一起使用，也就是说，它往往是表单中的一部分。为了使用方便，自定义一个名为 Wm_Plugin_Form_Element_Vcode 的表单元素类，并将其存放在 application\plugins\form\element 目录中，代码如下：

```
//Vcode.php 文件
class Wm_Plugin_Form_Element_Vcode extends Zend_Form_Element_Xhtml
{
    public $helper = 'formVcode';
}
```

从代码中可以看出，该类继承于 Zend_Form_Element_Xhtml 类，因而具有了 Zend_Form 表单的所有功能。另外，该代码还为“验证码”表单元素定义了一个名为 formVcode 的视图助手。

3. 添加验证码表单元素视图助手

打开 application\views\helpers 目录，添加新文件 FormVcode.php，创建名为 formVcode 视图助手，代码如下：

```
class Zend_View_Helper_FormVcode extends Zend_View_Helper_FormElement
{
  public function formVcode ($name, $value = null, $attribs = null)
  {
      $xhtml = '<input type="text" name="' . $name . '" id="' .
                  $name . '" maxlength="4"  value="' . $value . '" ';
      if (isset($attribs['textStyle'])) {
        $xhtml .= 'style="' . $attribs['textStyle'] . '" ';
      }
      if (isset($attribs['textClass'])) {
          $xhtml .= 'class="' . $attribs['textClass'] . '" ';
      }
      $xhtml .= '/>';
      $xhtml .= '<img id="vcodeImg" src="' . $this->view->baseUrl()
                                          . '/common/vcode" ';
      if (isset($attribs['imageStyle'])) {
          $xhtml .= 'style="' . $attribs['imageStyle'] . '" ';
      }
      if (isset($attribs['imageClass'])) {
          $xhtml .= 'class="' . $attribs['imageClass'] . '" ';
      }
      $xhtml .= '/>';
      $xhtml .= '<span ';
      if (isset($attribs['spanStyle'])) {
          $xhtml .= 'style="' . $attribs['spanStyle'] . '" ';
      }
      if (isset($attribs['spanClass'])) {
```

```
            $ xhtml . = 'class = "' . $ attribs[ 'spanClass'] . '" ';
        }
        $ xhtml . = ' />';
        $ xhtml . = '<a href = "javascript:' . $ attribs[ 'functionName']
                . '" class = "' . $ attribs[ 'aClass'] . '">看不清,换一张</a>';
        $ xhtml . = '</span>';
        return $ xhtml;
    }
}
```

上述阴影部分的代码表示验证码在页面中的图片显示。从这里可以看出,验证码来自于 Common 控制器的 vcode 方法。

4. 添加验证码表单元素验证器

前面在使用 Zend_Form 表单元素时都使用了一些验证器,例如非空验证、字符长度验证等,也可以为上述自定义的验证码表单元素添加验证器。

打开 application\plugins\validate 目录,添加新文件 VcodeRight. php,创建名为 VcodeRight 的表单元素验证器,代码如下:

```
class Wm_Plugin_Validate_VcodeRight extends Zend_Validate_Abstract
{
    //设置错误信息的键名
    const ERRORMESSAGE = 'notRight';
    //设置错误信息
    protected $ _messageTemplates = array(
        self::ERRORMESSAGE => '验证码输入错误!'
    );
    //实现接口的 isValid()方法,对输入数据进行验证
    public function isValid ( $ value)
    {
        $ this -> _setValue( $ value);
        $ sessionNamespace = new Zend_Session_Namespace( 'session');
        if (trim(strtolower( $ value)) !=
                strtolower( $ sessionNamespace -> validateCode)) {
            $ this -> _error(self::ERRORMESSAGE);
            return false;
        }
        return true;
    }
}
```

代码中的基类 Zend_Validate_Abstract 为抽象类,它实现了 Zend_Validate_Interface 接口,因而是一个 Zend Framework 的验证器,在验证时将用户输入的验证码与会话中存储的信息进行比对。

5. 添加验证码控制器方法

在 Common 控制器中添加 vcode 方法,代码如下:

```
public function vcodeAction()
{
        $ w = $ this -> getRequest() -> getParam( 'w');
```

```
        $h = $this->getRequest()->getParam('h');
        $f = $this->getRequest()->getParam('f');
        if ($w == null) {
            $w = 80;
        }
        if ($h == null) {
            $h = 35;
        }
        if ($f == null) {
            $f = 0;
        }
        $codeStr = "";
        $codeArray = array('0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z');
        for ($i = 0; $i < 4; $i ++) {
            $codeStr .= $codeArray[mt_rand(0,35)];
        }
        $sessionNamespace = new Zend_Session_Namespace('session');
        $sessionNamespace->validateCode = $codeStr;
      $validateCode = new Wm_Plugin_Util_ValidateCode($w, $h, $codeStr, $f);
        $validateCode->show();

        $this->_helper->layout->disableLayout();
        $this->_helper->viewRenderer->setNoRender();
    }
```

阴影部分代码完成验证码的输出与文本的保存。验证码文本保存在 session 会话空间中，以备在 Wm_Plugin_Validate_VcodeRight 验证器中与用户的输入进行比对。

6. 创建登录表单

使用前面章节中介绍的方法创建用户登录系统管理中心验证表单，代码如下：

```
class Wm_Form_AdminLogin extends Zend_Form
{
    public function init()
    {
        $this->setName('form_adminlogin')
             ->setMethod('post')
             ->addAttribs(array(
                 'style' => 'margin:0px; padding:0px'
             ));

             $this->addElements(array(
                    new Wm_Plugin_Form_Element_Vcode('vcode',array(
                           'required' => true,
                           'label' => '验证码：',
                           'attribs' => array(
                                  'textClass' => '',
                                  'textStyle' =>'position:relative;
                                    left:20px; top:0px; width:60px;
                                     height:18px; line-height:18px; ',
```

```
                                        'imageStyle'=>'position:relative;
                                                left:50px; top:15px;',
                                        'spanStyle'=> 'position:relative;
                                                top:25px; height:18px;',
                                        'aClass'=> 'a4',
                                     'functionName'=> 'changeValidateCode()'
                                ),
                                'filters'=> array('StringTrim'),
                                'validators'=> array(
                                        array('NotEmpty',true,
                                          array('messages'=> '请输入验证码')),
                                        new Wm_Plugin_Validate_VcodeRight()
                                          ),
                                'decorators'=> array(
                                        'ViewHelper',
                                        'Errors',
                                        array('HtmlTag',array('tag'=> 'td',
                                            'style'=> 'height:30px;')),
                                        array('Label',array('tag'=> 'th'))
                                )
                        )),
                        new Zend_Form_Element_Image('submitImage',array(
                                'required'=> false,
                                'label'=> '',
                                'src'=> '/images/btn_login.gif',
                                'attribs'=> array(
                                        'style'=> 'position:relative;
                                                left:20px; top:10px;'
                                ),
                                'decorators'=> array(
                                        'ViewHelper',
                                        array('HtmlTag',array('tag'=> 'td',
                                            'style'=> 'height:55px;')),
                                        array('Label',array('tag'=> 'th'))
                                )
                        ))
                    ));
                $this->setDecorators(array(
                        'FormElements',
                array('HtmlTag',array('tag'=>'table','class'=>'sheet')),
                        'form'
                ));
    }
}
```

这里调用 Zend_Form 的 addElements 方法通过直接创建表单元素对象来添加表单元素。其详细用法请大家参考第 4 章的有关内容。

阴影代码中的第 1 句使用前面自定义的“验证码”表单元素类 Wm_Plugin_Form_Element_Vcode 在表单中创建验证码,并用数组的方式添加相应的表单元素属性。阴影代码中的第 2 句定义了一个 JavaScript 函数名 changeValidateCode,它是用户单击验证码旁

边的“看不清，换一张”链接后的操作实现函数。

第 3 块阴影代码为“验证码”表单元素设置验证器，这里设置了两个验证器，一个是非空验证器 'NotEmpty'，这是 Zend Framework 预定义验证器，在前面的章节中已经多次使用，它的错误提示信息为“请输入验证码”；另一个验证器为类 Wm_Plugin_Validate_VcodeRight 的对象，即自定义的“验证码”验证器，它的错误提示信息为“验证码输入错误！”文本。

接下来的第 4 块阴影代码设置“验证码”表单元素装饰器。'ViewHelper'用我们自定义的 formVcode 视图助手来解析“验证码”表单元素；'Errors'解析验证器错误提示信息。最后的阴影部分代码用 Zend_Form 图片元素代替表单的“提交”按钮，以增强页面的效果。

7. 创建登录控制器方法

在 User 控制器中添加 adminlogin 方法，代码如下：

```
public function adminloginAction()
{
        $adminLogin = new Wm_Form_AdminLogin();
        if ($this->getRequest()->isPost()) {
            $formData = $this->getRequest()->getPost();
            if ($adminLogin->isValid($formData)) {
                ...
            }
        }

        $this->view->adminLoginForm = $adminLogin;
}
```

该方法输出登录表单，接收并处理用户的输入。阴影代码为表单验证，即调用上面 6 中设置的“非空”与“验证码输入错误”两个验证器。

8. 设计登录页面视图

打开系统管理中心登录视图文件 application\views\scripts\user\adminlogin.phtml，将原有代码修改为：

```
<br /><br />
<h1>后台管理中心登录验证</h1>
<hr /><br />
<?php echo $this->adminLoginForm ?>
<script>
function changeValidateCode(){
    var vcode = document.getElementById("vcodeImg");
    vcode.src = "<?php echo $this->baseUrl(
                    '/common/vcode/rand/" + Math.random() + "')?>";
}
</script>
```

注意：代码中的阴影部分，它们实现登录表单的显示与验证码的变换。

12.5.2 系统日志

系统日志即系统运行的工作记录。记录并读取系统日志文件，对于了解系统的运行状

态、出现的异常以及访问情况等都是很有必要的。在 Zend Framework 中提供了 Zend_Log 组件用于执行对系统日志文件的操作。

1. Zend_Log 组件

Zend_Log 是一个通用日志组件，该组件支持多个日志后端、格式化发送给日志的消息、过滤被记录的消息等。该组件包括以下 4 个对象：

1）Log 对象

Log 对象即 Zend_Log 类的实例，它是应用程序使用最多的对象。根据实际需要，可以创建任意多个 Log 对象，不同的 Log 对象之间不会相互影响。一个 Log 对象必须至少包含一个 Writer（容器）对象，还可以选择包含一个或多个 Filter（过滤器）对象。

2）Writer 对象

Writer 对象继承于 Log 容器抽象类 Zend_Log_Writer_Abstract，负责向存储容器中保存日志数据。常用的主要有 Zend_Log_Writer_Stream、Zend_Log_Writer_Null、Zend_Log_Writer_Db 以及 Zend_Log_Writer_Mock 对象等。

3）Filter 对象

Filter 对象实现了 Log 过滤器接口 Zend_Log_Filter_Interface，过滤被保存的日志数据。一个 Filter 对象可以应用到多个 Writer 对象中，或者在所有 Writer 之前应用一个 Log 对象，这样多个 Filter 过滤器对象可以串联起来。

4）Formatter 对象

Formatter 对象实现了 Log 数据格式化接口 Zend_Log_Formatter_Interface，负责在由 Writer 写入数据之前对日志数据的格式化。每一个 Writer 对象只能有一个 Formatter 格式化对象。

2. Log 对象的创建与使用

1）创建对象

如果要创建 Zend_Log 对象，只需要简单地使用 new 运算符调用 Zend_Log 类的构造方法即可。注意，一个 Zend_Log 对象必须至少有一个 Writer 对象，Writer 对象可以通过 Zend_Log 类的构造方法带入，也可以调用 Zend_Log 类的 addWriter 方法添加。例如：

```
$writer = new Zend_Log_Writer_Stream('php://output');
$log = new Zend_Log($writer);
```

或者：

```
$writer = new Zend_Log_Writer_Stream('php://output');
$log = new Zend_Log();
$log->addWriter($writer);
```

2）使用日志信息

Zend_Log 对象在创建成功之后就可以向其中添加日志信息了。日志信息的添加是通过 Zend_Log 类的 log 方法完成的，使用该方法还可以设置日志信息的类型。其语法格式如下：

```
log($message, $priority)
```

其中，参数 message 为需要记录的信息内容，以字符串表示；参数 priority 为信息类型，

也称为信息等级，用 Zend_Log 常量表示。Zend Framework 的内部日志等级有 8 种，分别为 EMERG、ALERT、CRIT、ERR、WARN、NOTICE、INFO 和 DEBUG。

用户除了可以使用 log 方法之外，还可以使用与消息等级同名的方法，且使用这些方法时不必给消息指定等级，消息会自动创建该等级的消息。例如：

```
$ writer = new Zend_Log_Writer_Stream('php://output');
$ log = new Zend_Log( $ writer);
$ log -> log('系统日志测试…',Zend_Log::INFO);
```

或者：

```
$ writer = new Zend_Log_Writer_Stream('php://output');
$ log = new Zend_Log( $ writer);
$ log -> info('系统日志测试…');
```

3）销毁日志对象

如果某个 Zend_Log 对象不再需要，可以将该对象销毁。如果要销毁一个 Zend_Log 对象，只需要为其赋值 Null 即可。在 Zend_Log 对象被销毁前会自动调用每个附加在 Log 上的 Writer 的 shutdown 方法，释放其所占的系统资源。

3. 系统日志的实现

下面以用户登录系统管理中心为例实现系统的日志功能。当用户通过图 12.4 所示的验证后，进入系统管理中心页面，此时，在系统日志文件中记录用户登录信息，包括登录时间、用户姓名、所属部门等。

在 User 控制器的 adminlogin 方法中添加如下代码：

```
public function adminloginAction()
{
        $ adminLogin = new Wm_Form_AdminLogin();
        if ( $ this -> getRequest() -> isPost()) {
            $ formData = $ this -> getRequest() -> getPost();
            if ( $ adminLogin -> isValid( $ formData)) {
                $ logDir = APPLICATION_PATH.'/logs/'.date('Y - m - d').'/';
                $ folder = new
                        Zend_Search_Lucene_Storage_Directory_Filesystem(
                         $ logDir);
                  $ fileName = $ logDir.'wm_log_login.txt';
                  $ file = fopen( $ fileName,'a');
                  $ writer = new Zend_Log_Writer_Stream( $ file);
                  $ log = new Zend_Log( $ writer);
                  $ str = $ this ->_user -> getIdentity() -> username;
                  $ str . = '**** '. $ this ->_user -> getIdentity() -> department;
                  $ str . = '**** 登录系统管理中心';
                  $ log -> log( $ str,Zend_Log::INFO);
                  $ this -> redirect('/admin/user/list');
            }
        }
        $ this -> view -> adminLoginForm = $ adminLogin;
    }
```

日志信息的存储方式(也就是 Log 的 Writer 对象)可以有多种类型,常用的有文件及数据库两种方式。这里采用文件的存储形式,文件名为 wm_log_login.txt,位于 application\logs\year-month-day\目录下。

代码首先创建日志目录及文件,然后以追加的方式打开日志文件;接着创建 Log 对象及其 Writer 容器对象;最后将日志信息写入日志文件中,效果如图 12.5 所示。

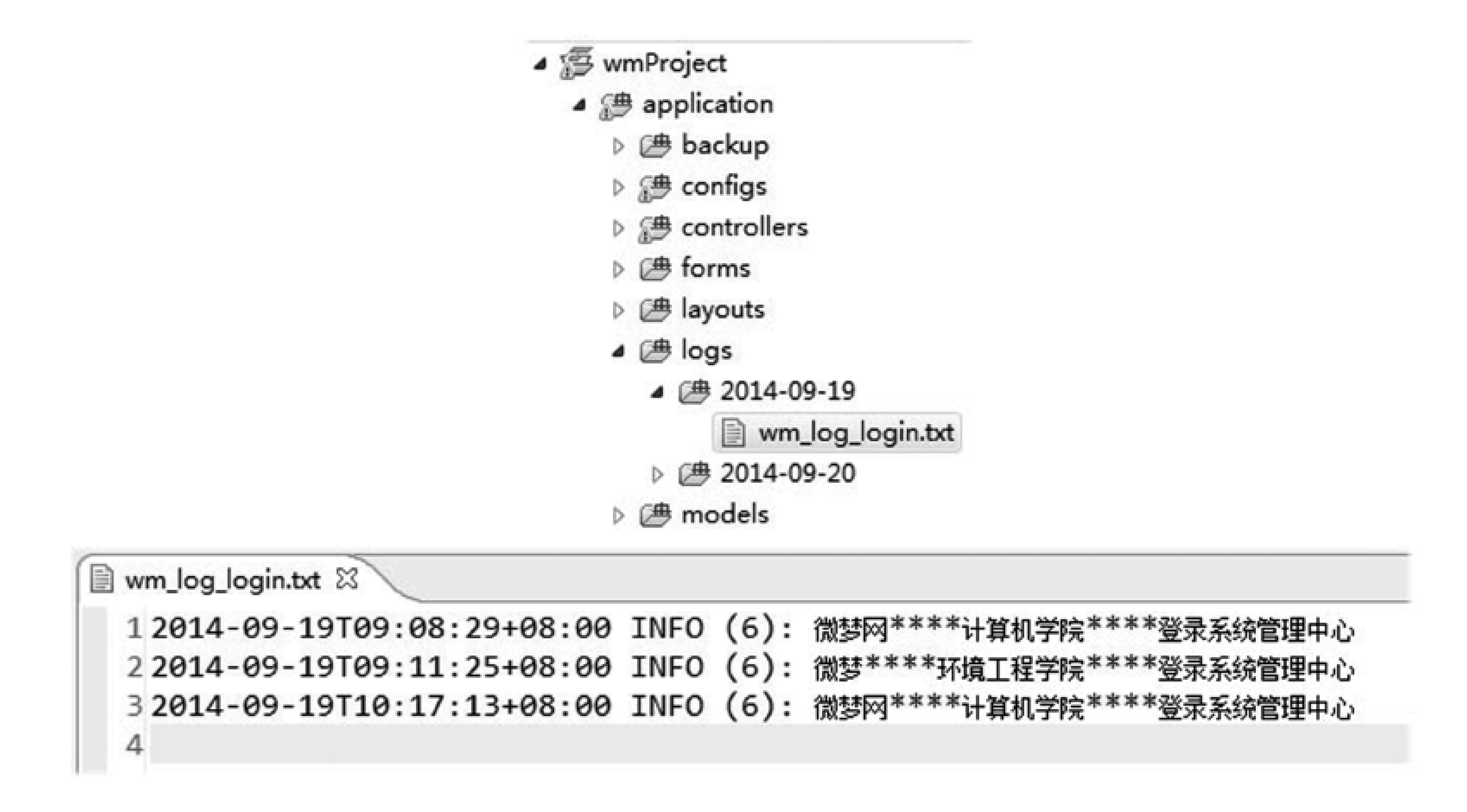


图 12.5　日志目录结构及文件详情

12.5.3　数据备份

对于 Web 应用来说,数据备份的重要性是毋庸置疑的。数据库的备份一般包括完全备份、事务日志备份、差异备份与文件备份 4 种类型,用户应根据具体情况选用。

数据库备份的方式有很多,可以直接采用数据库的命令形式,也可以采用 PHP 编程的方式。下面简单介绍以 PHP 编程的方式备份整个数据库及下载单个数据表数据的方法。

1. 设计视图

使用前面章节中介绍的方法在 admin 模块中创建 Backup 控制器,并在其中添加 dbbackup、dbdownload 及 store 方法,编写视图文件代码。

1) index.phtml 文件

```
<br /><br />
<p>系统数据备份</p>
<hr />
<p><a href = "/admin/backup/dbbackup">数据库备份</a>  对系统数据库进行备份,文件格式为 sql </p>
<hr />
<p><a href = "/admin/backup/dbdownload">数据表下载</a>  对系统数据库中的数据下载备份,文件格式为 txt </p>
<hr />
```

当用户在系统后台管理中心中单击左侧快捷菜单中的“其他信息管理”|“数据备份”菜单项后,进入如上代码所示的 index.phtml 主页面。该页面中设置了“数据库备份”与“数据表下载”两个超级链接,分别完成数据库整体备份与部分数据下载备份功能。

2）dbbackup. phtml 文件

```
<br /><br />
<p>数据库备份</p>
<hr /><br />
<form method="POST" action="">
<input type="text" name="filename" value="<?php echo 'Db_'.date('Ymd').'.sql'?>" />
    <input type="submit" value="备份" />
</form>
```

在上述“数据库备份”页面代码中设置了一个表单来接收数据库备份文件名，这里默认为“Db_年月日.sql”。

3）dbdownload. phtml 文件

```
<br /><br />
<p>下载数据表</p>
<hr /><br />
<form method="POST" action="">
<select name="table">
  <?php foreach ( $this->dbtables as $arry_table){
      foreach ( $arry_table as $key=>$table){
  ?>
      <option value="<?php print( $table); ?>">
          <?php print( $this->tableName( $table)); ?></option>
  <?php }} ?>
</select>
    <input type="submit" value="下载" />
    <input type="reset"  value="取消" />
</form>
```

在用户下载数据表中的数据时必须确定数据表的名称，这里通过 HTML 的“选择”表单元素来完成该项功能，如图 12.6 所示。选择列表中的数据表名来自于 Backup 控制器的 dbdowmload 方法，列表中的文本来自于 tableName 视图助手，其代码如下：

```
class Zend_View_Helper_TableName
{
    public function TableName ( $name)
    {
        $tableName = '';
        switch ( $name) {
            case 'tb_docs':
            $tableName = '公文数据';break;
            case 'tb_users':
            $tableName = '用户数据';break;
            case 'tb_affair':
            $tableName = '事务数据';break;
            case 'tb_userfile':
            $tableName = '用户文件';break;
            default:
                $tableName = '…';break;
        }
```

```
            return $tableName;
        }
    }
```

该视图助手通过数据表名获取列表项的描述文本，即用户需要下载的数据类型。对于视图助手的定义请读者参考第 10 章中的相关内容。

创建完 M-V-C 各部件后，就可以在浏览器中预览页面效果了，如图 12.6 所示。该图中左、中、右部分分别为 index.phtml、dbbackup.phtml、dbdownload.phtml 页面。



图 12.6　数据备份页面效果

2. 实现数据库备份

在 Backup 控制器中初始化一个数据库适配器，并在 dbbackup 方法中添加代码：

```
class Admin_BackupController extends Zend_Controller_Action
{
    protected $_db = null;
    public function init()
    {
        $this->_db = Zend_Db_Table_Abstract::getDefaultAdapter();
        $this->_helper->layout()->setLayout('admin');
    }
...
public function dbbackupAction()
{
      if($this->getRequest()->isPost()){
      $filename = $this->getRequest()->getParam('filename');
      $backupDir = APPLICATION_PATH.'/backup/';
$folder = new Zend_Search_Lucene_Storage_Directory_Filesystem($backupDir);
      $sql = 'd:\xampp\MySQL\bin\mysqldump -h127.0.0.1
              -uroot -p123456 db_wmoams > '.$backupDir.$filename;
      exec($sql);
      echo "<script>alert('备份成功');
                    location = '/admin/backup'</script>";
        }
    }
```

```
}
```

如果要对数据库进行备份，必须获取数据库适配器。本系统的数据库适配器是在配置文件中设置的，这里通过 Zend_Db_Table_Abstract 类的静态方法 getDefaultAdapter 得到。

在 PHP 中备份数据库，通过调用 exec()函数执行系统命令来完成。代码中的参数为作者本地环境参数，在实际应用时应适当调整。

3. 下载数据表数据

在 Backup 控制器的 dbdownload 方法中添加代码：

```
public function dbdownloadAction()
{
        $tableObj = $this->_db->query('SHOW TABLES');
        $tables = $tableObj->fetchAll();
          if($this->getRequest()->isPost()){
               $tableName = $this->getRequest()->getParam('table');
               $filename = $tableName.date('Ymd').'.txt';
               $this->redirect('/admin/backup/store/tableName/'
.$tableName.'/fileName/'.$filename);
          }
          $this->view->dbtables = $tables;
     }
     public function storeAction()
     {
          $fileName = $this->getRequest()->getParam('fileName');
          $tableName = $this->getRequest()->getParam('tableName');
          header('Content-Type: application/octet-stream');
          header('Content-Disposition: attachment; filename='.$fileName);
          $select = new Zend_Db_Select($this->_db);
          $select->from($tableName);
          $rs = $select->query();
          for($i = 0; $i < $rs->columnCount(); $i++){
               $data = $rs->getColumnMeta($i);
               print($data['name']."\t");
          }
          print("\n");
          $row = $rs->fetch(PDO::FETCH_NUM);
          while($row){
               for($i = 0; $i < $rs->columnCount(); $i++){
                    print(mb_convert_encoding($row[$i],
                          'gb2312','UTF-8')."\t");
               }
               $row = $rs->fetch(PDO::FETCH_NUM);
               print("\n");
          }
               $this->_helper->layout->disableLayout();
               $this->_helper->viewRenderer->setNoRender();
     }
```

这里将下载页面的显示与数据的下载分别用两个不同的方法实现。代码中的第 1 句阴影部分通过数据库适配器查询得到系统数据库的所有数据表；第 2 块阴影部分设置下载页面类型及备份文件名称；第 3 句阴影代码获取下载数据表数据；第 4 句阴影代码将数据表

字段名写入文件中；最后第 5 句阴影代码写入文件的是数据表中的数据记录。

12.6 本章小结

本章实现办公自动化管理系统的访问控制、缓存等 Web 应用的安全、性能优化技术方案，并对系统进行了简要的完善，包括图形验证码、系统日志、数据备份等。Zend Framework 应用的访问控制以及数据和页面的缓存由 Zend_Acl 及 Zend_Cache 组件完成。

本章重点掌握 Zend Framework 框架的角色、资源、控制列表等基本概念，访问控制规则的设置以及使用插件实现访问控制的方法；对于 Zend_Cache 缓存组件，重点掌握其工作原理以及前端、后端的概念和相关类方法的使用。

参 考 文 献

[1] [澳]威利，汤姆森. PHP 和 MySQL Web 开发[M]. 北京：机械工业出版社，2009.
[2] [英]道尔. PHP 5. 3 入门精典[M]. 吴文国，黄海降译. 北京：清华大学出版社，2010.
[3] 陈营辉，赵伟，赵海波，等. Zend Framework 技术大全[M]. 北京：化学工业出版社，2010.
[4] 欧雪冰. PHP 顶级框架 Zend Framework 开发实战[M]. 北京：电子工业出版社，2012.
[5] 明日科技，刘中华，潘凯华，等. PHP 项目开发案例全程实录[M]. 2 版. 北京：清华大学出版社，2011.
[6] 谭贞军. 深入体验 PHP 项目开发[M]. 北京：清华大学出版社，2011.
[7] 王志刚，朱蕾. PHP5 应用实例详解：使用 Zend Framework & Smarty 构筑真正的 MVC 模式应用[M]. 北京：电子工业出版社，2010.
[8] [美]Jason Lengstorf. 深入 PHP 与 jQuery 开发[M]. 魏忠译. 北京：人民邮电出版社，2011.
[9] 马石安，魏文平. 数据结构与应用教程(C++版)[M]. 北京：清华大学出版社，2012.
[10] 马石安，魏文平. 面向对象程序设计教程(C++语言描述)[M]. 2 版. 北京：清华大学出版社，2014.